Titre: Title:	Identification de méthodes pour l'évaluation des grammaires de langues naturelles
Auteur: Author:	Dominic Letarte
Date:	2006
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Letarte, D. (2006). Identification de méthodes pour l'évaluation des grammaires de langues naturelles [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7722/

UNIVERSITÉ DE MONTRÉAL

IDENTIFICATION DE MÉTHODES POUR L'ÉVALUATION DES GRAMMAIRES DE LANGUES NATURELLES

DOMINIC LETARTE
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
AVRIL 2006

Your file Votre référence
ISBN: 978-0-494-17953-6
Our file Notre référence
ISBN: 978-0-494-17953-6

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

IDENTIFICATION DE MÉTHODES POUR L'ÉVALUATION DES GRAMMAIRES DE LANGUES NATURELLES

présenté par: LETARTE Dominic

en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

M. GALINIER Philippe, Doct., président

M. GAGNON Michel, Ph.D., membre et directeur de recherche

M. MERLO Ettore, Ph.D., membre et codirecteur de recherche

Mme BOUCHENEB Hanifa, Doctorat, membre

Ce travail est dédié à tous ceux que j'ai oublié parce que
j'étais en retard pour finir quelque chose.

REMERCIEMENTS

Merci à Gilbert Babin, Jian-Yun Nie, Guy Lapalme, Philippe Langlais, Michel Gagnon et Ettore Merlo qui m'ont guidés, encouragés et supportés pendant mon cheminement universitaire.

RÉSUMÉ

Le développement des grammaires des langues naturelles est une tâche longue et difficile mais qui pourrait être facilité en utilisant des outils de développement plus modernes. Nous proposons d'utiliser une technique issue de la théorie des compilateurs, soit l'analyse statique du code source. Nous proposons d'utiliser cette technique pour déterminer les valeurs possibles des variables dans les grammaires de langues naturelles afin de produire des messages d'avertissements utiles pour le développeur. Nous avons implémenté cette nouvelle analyse et nous l'avons appliquée avec succès sur une petite grammaire du portugais.

ABSTRACT

The construction of grammars for natural language parsing is a task that is long and difficult. But this task can be facilitated by the usage of more modern development tools. We propose to use a technique coming from the theory of compilers : static analysis of source code. We propose to use that technique for determining the possibles values of variables used in grammars for natural language processing, this will permit us to generate warnings messages useful for the developer of the grammar. We have implemented this new analysis et we have used it with success on a small grammar of Portuguese.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

BNF	Backus Naur Form
def-origin	Le couple définition et origine (d'une valeur)
EBNF	Extended Backus Naur Form
HPSG	Head Phrase Structure Grammar
LHS	Left Hand Side, la partie de gauche d'une règle de grammaire
NLP	Natural Language Processing
RHS	Right Hand Side, la partie de droite d'une règle de grammaire

LISTE DES ANNEXES

INTRODUCTION

Domaine d'étude

Les grammaires de langues naturelles sont utilisées dans plusieurs outils de traitement de la langue naturelle, principalement les logiciels de correction grammaticale et les logiciels de traduction automatique. Dans une certaine mesure, elles sont aussi utilisées par les logiciels de reconnaissance vocale et les moteurs de recherche sur le web. Les grammaires sont utilisées afin de produire une analyse du texte. Cette analyse permet de lever les ambigüités sur la catégorie grammaticale des mots. Par exemple, déterminer si le mot "marche" qui apparait dans une phrase est un nom commun, comme dans "monter sur la marche" ou s'il est un verbe, comme dans "je marche vers la sortie". L'analyse grammaticale permet aussi de déterminer la fonction des mots dans la phrase. Par exemple, dans "Annie donne un livre à Charles.", l'analyse grammaticale permet de déterminer que Annie est le sujet de la phrase et que Charles est un complément. Ces constatations sont triviales avec des phrases simples et courtes mais deviennent extrêmement complexes avec des phrases plus longues.

Le fonctionnement de ces analyseurs de la langue naturelle repose sur une grammaire. Cette grammaire est comparable aux grammaires BNF que l'on utilise couramment pour reconnaître les langages informatiques. Cependant, elles sont beaucoup plus volumineuses et utilisent des formalismes particuliers qui permettent de mieux gérer cette taille ainsi que diverses autres particularités des langages naturels.

Problématique

Les outils pour développer des grammaires sont archaïques. Si des formalismes spécifiques au traitement des langues naturelles ont été développés, il manque toujours

les autres outils nécessaires dans le processus de développement. On développe les grammaires de la même manière que l'on développait un logiciel il y a trente ans. Le développement des grammaires se résume à la séquence suivante : écrire des règles dans un éditeur de texte ; ne pas oublier de sauvegarder le fichier ; démarrer un analyseur syntaxique en chargeant ce fichier de règles ; espérer qu'il n'y a pas d'erreurs de syntaxe dans le fichier de règles ; analyser une phrase ; et observer si le résultat est celui attendu. Si le résultat n'est pas celui attendu, il est souvent possible de générer une trace d'analyse (l'équivalent de mettre des "print" un peu partout dans le programme) ou de mettre une partie de la grammaire en commentaire afin de cerner le problème. Cette technique de développement est identique à celle employée pour développer un programme informatique lorsque nos seuls outils sont un compilateur et un éditeur de texte.

Des outils plus modernes sont disponibles pour développer des programmes informatiques. Les compilateurs génèrent des messages d'avertissement plus utiles que seulement des erreurs de syntaxe et l'usage des débogueurs est une chose courante. Des outils plus spécialisés sont aussi disponibles comme les optimisations automatiques du code, la détection des fuites de mémoire, la détection des clones et la génération automatique de tests. Ces outils reposent sur les développements de la théorie des compilateurs plus spécialement l'analyse statique et dynamique des programmes. Ces outils ont surtout été appliqués à l'analyse des programmes en langages impératifs comme Cobol, C et Java. Bien que moins répandus, ils existent aussi pour les langages symboliques comme Lisp et Prolog.

Les grammaires, qu'elles soient BNF ou spécialisées pour le traitement de la langue, sont aussi des langages logiques dans la même famille que Prolog. En effet, on peut souvent directement traduire les formalismes de grammaires, y compris BNF, en Prolog sans aucunes pertes. Il est possible d'utiliser des outils modernes de développement avec des langages de programmation logique. Pourtant ces outils n'ont pas été adaptés

pour le développement des grammaires. Une des raisons est peut-être que les grammaires pour les langages informatiques sont souvent d'une petite taille et peuvent être développées par une seule personne en utilisant peu d'outils. Par exemple des langages considérés simples, comme Cobol, C ou Pascal, peuvent souvent être exprimés en moins d'une centaine de règles et les langages plus complexes comme C++ ou Java peuvent atteindre un peu moins de trois cents règles. Comparativement les grammaires pour les langues naturelles sont plus souvent développées par une équipe et il n'est pas rare de rencontrer des grammaires de plus de mille règles. Des outils de développement modernes devraient être essentiels pour développer et entretenir ce type de grammaires.

Thèse

Nous croyons qu'il est possible d'utiliser des outils modernes de développement de logiciels basés sur l'analyse statique pour développer des grammaires. Ces outils seront plus particulièrement utiles pour les grammaires des langues naturelles qui sont plus volumineuses et plus difficiles à développer et à entretenir. Nous croyons que l'usage de ces outils permettra des générer des informations utiles pour les développeurs ce qui permettra d'augmenter la qualité des grammaires produites, d'accélérer la vitesse de leur développement et de faciliter l'entretien de ces grammaires.

Expérimentation

Comme la production d'une suite entière d'outils pour le développement des grammaires est une lourde tâche, nous nous sommes concentrés sur l'implantation d'une seule analyse qui nous semblait la plus prometteuse et la plus simple. Nous avons implanté une nouvelle analyse statique du code, soit l'analyse des valeurs possibles

des variables dans une grammaire. Cette analyse est aussi décrite dans Merlo *et al.* (2004) et dans Gagnon *et al.* (2004). Les résultats de cette analyse permettent de générer des messages d'avertissements sur les valeurs possibles (ou impossibles) des variables. L'analyse des résultats permet aussi de générer d'autres informations utiles pour la documentation des grammaires.

Voici l'organisation de ce mémoire. Les trois premiers chapitres présentent la revue bibliographique, respectivement le formalisme de grammaire utilisé, les méthodes d'évaluation actuelles et l'analyse statique. Viennent ensuite, dans le chapitre 4, deux variations du modèle d'analyse statique que nous avons développés pour résoudre le problème présenté. Le chapitre 5 décrit *Glint*, l'outil que nous avons développé et qui implémente les deux modèles décrits. Le deux dernier chapitre décrit et discute les résultats obtenus. Finalement les annexes contiennent des exemples des fichiers de sortie de Glint ainsi que des saisies d'écran de son intégration à Emacs et à Developper Studio.

CHAPITRE 1

LES GRAMMAIRES

Ce chapitre présente les notions de base sur les grammaires et les outils de parsage pour les langues naturelles. La première section présente les grammaires formelles et la hiérarchie des langages de Chomsky. La section suivante donne un exemple d'une grammaire classique libre de contexte en format BNF. La dernière section présente le formalisme de grammaires syntagmatique, qui est un formalisme de grammaires spécialisé pour le traitement de la langue naturelle, et basé sur un ensemble de contraintes régies par un mécanisme d'unification (on les désigne aussi par le terme "grammaire d'unification").

1.1 Les grammaires

Les grammaires formelles sont utilisées en linguistique et en informatique. Elles servent à décrire quelles séquences de symboles seront reconnues comme valides dans un certain langage. Par exemple, une grammaire formelle pour le français contiendra comme symboles tous les mots du dictionnaire. La grammaire contiendra aussi un ensemble de règles dans un format particulier. Ces règles permettront de reconnaitre que la séquence : *je mange une pomme* est une phrase valide en français tandis que *manger pomme je* ne l'est pas. C'est la même technique qui est utilisée pour la compilation des langages informatique. Les symboles sont tous les mots réservés du langage, tels les identificateurs, les opérateurs, etc. Par exemple, une grammaire du langage *C* permet de dire que x *+= abs(y) + 2;* est une instruction valide en C tandis que x = *5 * (2 +* x*;* est invalide (il manque la parenthèse fermante).

Les grammaires ont été décrites du point de vue linguistique dans Chomsky (1959) et d'un point du vue plus informatique dans Aho *et al.* (1986). Nous résumons rapidement en rappelant que les grammaires sont décrites formellement par

$$G = (T, N, S, REGLES) \tag{1.1}$$

tel que G est la grammaire à décrire, T est l'ensemble des symboles terminaux pouvant apparaître dans la chaîne à reconnaitre, N est l'ensemble des symboles non-terminaux permis dans les règles de la grammaire, S est le symbole de départ (*Start*) de la grammaire tandis que $REGLES$ est l'ensemble des règles de la grammaire. Les symboles terminaux sont ceux que l'on retrouve dans le dictionnaire du langage ainsi que dans la séquence à analyser alors que les non-terminaux sont des symboles "intermédiaires" que l'on pourra utiliser seulement dans les règles de la grammaire. La composition des règles de grammaire dépend du type de grammaire à produire. La Figure 1.1 montre un exemple d'une grammaire.

$N = \{D, NP, VP, Det, N, V\}$
$T = \{the, baby, slept\}$
$S = S$

$$REGLES = \left\{ \begin{array}{rcl} S & \rightarrow & NP, VP \\ NP & \rightarrow & Det, N \\ VP & \rightarrow & V \\ & & \\ Det & \rightarrow & the \\ N & \rightarrow & baby \\ V & \rightarrow & slept \end{array} \right\}$$

FIGURE 1.1 Exemple d'une grammaire BNF pour un langage avec trois mots

Les langages reconnaissables par une grammaire formelle ont été classifiés par Chomsky (1959) selon le niveau de sophistication requis par le parseur pour reconnaître ce langage, comme illustré dans la Figure 1.2. Les langages les plus simples, les langages réguliers, peuvent être reconnus par un automate déterministe utilisant peu de

mémoire et en temps linéaire en fonction de la longueur du texte à reconnaître. Les langages les plus complexes nécessitent des algorithmes beaucoup plus complexes qui s'exécutent en temps requis et en consommation de mémoire quadratique.

Hierarchie de Chomsky	Grammaire	Langage	Automate minimal
Type-0	sans restrictions	récursif	machine de Turing
Type-1	sensible au contexte	sensible au contexte	linear-bounded
Type-2	libre de contexte	libre de contexte	à pile
Type-3	régulière	régulier	état finis

FIGURE 1.2 Chomsky Grammars Hierarchy. Each category of languages or grammars is a proper superset of the category directly beneath it.

La question de savoir quel est le niveau de complexité de la langue naturelle est plus difficile qu'on peut l'imaginer. La question *Est-ce que la langue naturelle est libre de contexte ?* a des implications profondes dans certaines théories linguistiques au niveau de la représentation mentale du langage. La réponse est vraiment plus du domaine de la philosophie pour le moment (voir Shieber (1985)). La réponse la plus répandue et la plus pratique pour les informaticiens est "la langue naturelle est faiblement contextuelle", tel que décrit par Huybregts (1984). C'est à dire que la langue naturelle est majoritairement libre de contexte sauf pour quelques rares exceptions, donc en pratique on la traite comme un langage libre de contexte. Par contre, on est conscient qu'il pourrait y avoir des exceptions difficilement traitables. Ce qui n'empêche pas que la majorité des grammaires existantes pour les langues naturelles sont toutes des grammaires libres de contexte.

Par exemple, la grammaire de la Figure 1.1 peut être utilisée pour analyser la phrase *the baby slept.* Le résultat de l'analyse est illustré par la Figure 1.3.

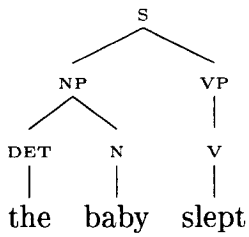

FIGURE 1.3 Analyse de la phrase *the baby slept* avec la grammaire de la Figure 1.1

D	→	Np-sing,Vp-sing
D	→	Np-plur,Vp-plur
Np-sing	→	Det-sing,N-sing
Np-plur	→	Det-plur,N-plur
Vp-sing	→	V-sing
Vp-plur	→	V-plur
Det-sing	→	le
Det-plur	→	les
N-sing	→	bébé
N-plur	→	bébés
V-sing	→	dors
V-plur	→	dorment

FIGURE 1.4 Grammaire BNF traitant le nombre (singulier ou plurierl)

1.2 Les grammaires d'unifications

Si on voulait raffiner la grammaire de la Figure 1.1 afin d'y intégrer les règles d'accords pour le singulier et le pluriel, on devrait remplacer les symboles non-terminaux par des symboles plus précis N-sing, N-plur, Det-sing, Det-plur, Np-sing, Np-plur, etc. Le nombre de règles sera doublé comme dans la Figure 1.4 Si on veut ajouter le traitement des accords en genre en plus des accords en nombre, les catégories pour les noms seulement deviendront : N-fem-sing, N-masc-sing, N-fem-plur, N-masc-plur. Encore une multiplication par deux. Le nombre de symboles non-terminaux et de règles explose rapidement avec le nombre d'attributs que l'on veut encoder avec les catégories grammaticales. Il est courant en traitement de la langue de vouloir encoder plusieurs dizaines d'attributs différents. Les grammaires BNF deviennent rapidement impraticables.

Les structures de traits des grammaires d'unification permettent d'éviter le problème de l'explosion du nombre de règles lorsque l'on raffine les catégories. Une grammaire d'unification est un formalisme de grammaires libres de contexte où les symboles non-terminaux sont remplacés par des structures de traits. Ces structures permettent d'augmenter l'expressivité des règles de grammaire et de produire beaucoup plus simplement des règles de grammaire complexes.

Les structures de traits ont étés décrites par Martin Kay dans sa grammaire fonctionnelle *Functional grammar*, Kay (1979), puis mises en pratique entre autres dans *The Core Language Engine* de Alshawi (1992), et dans les grammaires HPSG (*Head-Driven Phrase Structure Grammar*) de Pollard et Sag (1994).

Par exemple, la structure de la Figure 1.5 indique que la valeur du trait *categorie* est *nom*, que celle de *genre* est *masc* (masculin) et que celle de *nombre* est *plur* (pluriel).

On peut aussi utiliser des variables dans les structures de traits. Dans ce document,

$$\begin{bmatrix} \text{categorie} & \text{nom} \\ \text{genre} & \text{masc} \\ \text{nombre} & \text{plur} \end{bmatrix}$$

FIGURE 1.5 Exemple d'une structure de traits

nous utiliserons toujours des identificateurs commençant par une minuscule pour les noms de traits et les valeurs tandis que les identificateurs commençant par une majuscule seront réservés pour les variables.

Règle d'unification :

$$\begin{bmatrix} \text{cat} & \text{np} \\ \text{nombre} & \text{X} \\ \text{genre} & \text{Y} \end{bmatrix} \rightarrow \begin{bmatrix} \text{cat} & \text{det} \\ \text{nombre} & \text{X} \\ \text{genre} & \text{Y} \end{bmatrix}, \begin{bmatrix} \text{cat} & \text{n} \\ \text{nombre} & \text{X} \\ \text{genre} & \text{Y} \end{bmatrix}$$

Équivalent BNF :

$$\begin{aligned} \text{Np-masc-sing} &\rightarrow \text{Det-masc-sing,N-masc-sing} \\ \text{Np-masc-plur} &\rightarrow \text{Det-masc-plur,N-masc-plur} \\ \text{Np-fem-sing} &\rightarrow \text{Det-fem-sing,N-fem-sing} \\ \text{Np-fem-plur} &\rightarrow \text{Det-fem-plur,N-fem-plur} \end{aligned}$$

FIGURE 1.6 Une règle d'unification et son équivalent BNF

La règle de la Figure 1.6 est la règle traitant le cas $np \rightarrow det, n$ mais en précisant par la variable X dans la partie de droite que le trait *nombre* du déterminant (det) devra avoir la même valeur que le trait *nombre* du nom (n). L'emploi de la variable X dans la partie de gauche de la règle permet de préciser que le non-terminal produit (NP) possédera lui aussi le même nombre. On peut observer comment cettre règle se développerait en BNF.

La grammaire illustrée par la figure 1.7 donne un exemple plus complet. Cette dernière permet de produire la dérivation illustrée à la figure 1.8.

La présence d'attributs, de valeurs et de variables complique le travail du parseur. Il est plus difficile de déterminer si le résultat produit par la partie de droite d'une règle est compatible avec la partie de gauche d'une autre règle. La solution à ce problème est l'unification. L'unification est un algorithme qui permet de déterminer si deux expressions contenant des variables peuvent être rendues identiques en remplaçant les variables par des expressions. Si les deux expressions peuvent être rendues identiques, l'unification réussira et les valeurs des variables seront restreintes aux valeurs qui permettent à l'unification de réussir.

$$
\begin{aligned}
S &\rightarrow NP[\ \textsc{num}\ X\ ],\ VP[\ \textsc{num}\ X\ ] \\
NP[\ \textsc{num}\ X\ ] &\rightarrow DET[\ \textsc{num}\ X\ ],\ N[\ \textsc{num}\ X\ ] \\
VP[\ \textsc{num}\ X\ ] &\rightarrow V\left[\begin{array}{ll}\textsc{num} & X \\ \textsc{val} & \text{intr}\end{array}\right] \\
DET[\ \textsc{num}\ \text{sing}\ ] &\rightarrow \text{the} \\
N[\ \textsc{num}\ \text{sing}\ ] &\rightarrow \text{baby} \\
V\left[\begin{array}{ll}\textsc{num} & \text{sing} \\ \textsc{val} & \text{intr}\end{array}\right] &\rightarrow \text{slept}
\end{aligned}
$$

FIGURE 1.7 Exemple de dérivation

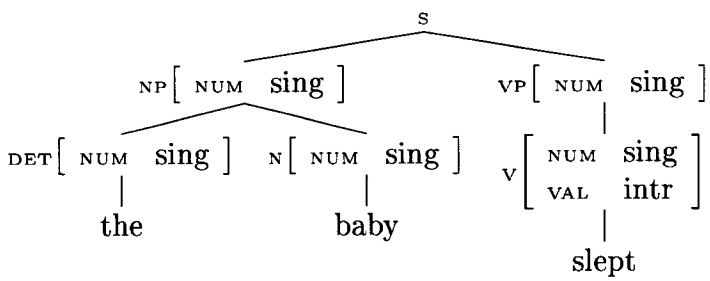

FIGURE 1.8 Example of derivation

CHAPITRE 2

L'ÉVALUATION DES GRAMMAIRES

Ce chapitre aurait pu s'écrire d'une manière simple : nous proposons une nouvelle approche à l'évaluation des grammaires et nous n'avons pas trouvé de précédents dans la littérature. Plutôt, nous présentons ici tout ce qui est englobé par le terme évaluation dans le domaine des systèmes à base de grammaires. Notre approche sera descriptive et les systèmes seront présentés brièvement. Ceci permettra au lecteur de connaître les systèmes qui pourront être comparés avec le notre. Nous n'aborderont pas le fonctionnement détaillé des autres systèmes et nous ne tenterons pas de trouver quel est le meilleur ou le moins bon. Ces systèmes ont été conçus pour évaluer les grammaires selon leurs propres critères différents des nôtres. Il existe beaucoup de travaux qui comparent les différents systèmes d'évaluations et qui essaient de déterminer le meilleur. Les travaux référés dans la prochaine section sur la classification des méthodes d'évaluations peuvent être consultés pour avoir une idée de la diversité des opinions sur le sujet de la meilleure méthode d'évaluation. Nous ne pouvons nous empêcher de remarquer que la littérature sur le classement et l'évaluation des systèmes d'évaluation est abondante. Par contre le nombre de système à classer et à évaluer est beaucoup plus restreint.

L'évaluation des grammaires de langues naturelles et des systèmes de traitement des langues naturelles est un sujet d'importance pour les organismes subventionnaires, les chercheurs et les utilisateurs de ces technologies. Plusieurs publications s'y sont intéressées. On doit d'abord noter la série de conférences LREC[1] (Language Resources and Evaluation Conferences), puis la série de Workshop IWPT[2] (International Workshop

[1] `http://www.lrec-conf.org`

[2] `http://bulba.sdsu.edu/iwpt05/` et `http://hmi.ewi.utwente.nl/sigparse/`

on Parsing Technology) et le Workshop on Evaluation Initiatives in Natural Language Processing [3] de EACL 2003 (European Chapter of the Association for Computational Linguistics). Les articles suivants font un tour d'horizon des méthodes d'évaluations : King (1996), Carroll *et al.* (1997), Bangalore *et al.* (1998) et Hirschman et Mani (2003).

Malgré toute cette attention pour le domaine, les consensus et les normes d'évaluations reconnues sont plutôt rares. Le consensus est plutôt qu'il faut se méfier des méthodes qui permettent d'évaluer la qualité d'un système ou bien de comparer des systèmes entre eux. Une citation classique dans le domaine de l'évaluation des grammaires est celle de John Carroll (1994) dans un article où il démontre que les mesures de complexité théorique ne sont pas fiables pour prédire le comportement des grammaires. Les résultats impliquent que l'étude et l'optimisation des parseurs basés sur l'unification doivent reposer sur des données empiriques jusqu'à ce que la théorie des mesures de complexités puisse prédire plus précisément le comportement de ces grammaires. Une recherche dans les bases de données bibliographiques montre que cet article est toujours d'actualité et on l'utilise souvent pour justifier des résultats empiriques lorsqu'une démonstration théorique est difficile ou impossible.

D'autres ne se limitent pas à souligner les lacunes de la théorie de la complexité des algorithmes. Karen Sparck Jones et Julia R. Galliers (1996) ont produit un rapport qui a été abondamment cité puis repris dans un livre où elles montrent que les méthodes d'évaluations sont très sensibles à des facteurs externes au système et qu'il est difficile de produire des métriques qui ne sont pas biaisées. En évaluation, il est essentiel de toujours observer les facteurs environnementaux. L'implication de cela est que même si l'évaluation est effectuée correctement, elle est d'une valeur limité : ce qui compte c'est le système et son environnement. Leurs travaux ont identifiés beaucoup d'évaluations qui ont étés faites sur plusieurs systèmes et beaucoup de lacunes dans les évaluations

[3] `http://www.dcs.shef.ac.uk/~katerina/EACL03-eval/`

jusqu'au point de dire que les évaluations sont inutiles. La position qu'ils défendent est qu'il faut plus se préoccuper de savoir si les usagers sont satisfaits du système.

Plus récemment, Diego Mollá (2003) à de nouveau confirmé cette opinion lorsqu'il a effectué l'évaluation des deux grammaires *Link Grammar* et *Conexor Functional Dependency Grammar*. Ces évaluations renforcent la position de Galliers et Sparck (1993) que les mesures intrinsèques sont d'une valeur très limités. En particulier, leurs évaluations du fonctionnement interne de ces systèmes ont produits des résultats qui sont en contradiction avec l'usage le plus intuitif de ces systèmes. Leur travail montre que le système qui a la meilleure évaluation n'est souvent pas le meilleur.

Une explication à toute cette suspicion envers les méthodes formelles pour évaluer les systèmes de traitement de la langues est avancée par Margaret King (1996). Ce rejet des méthodes d'évaluations formelles par la plupart des spécialistes du domaine serait dû au rapport ALPAC (1966). La controverse soulevée par cette évaluation fut immense, à la hauteur des conséquences qui ont suivi, soit l'arrêt total du financement des activités en traitement de la langue au États-Unis pendant les vingt années suivantes. L'effet fut moins drastique dans le reste du monde mais le rythme des travaux fut considérablement ralenti tout de même. Pourtant ALPAC était une des premières évaluations vraiment sérieuses et rigoureuses des systèmes de traitement de la langue. Néanmoins, on lui reproche toujours d'être une commande politique et d'avoir utilisé une méthode d'évaluation délibérément biaisée. Les conclusions du rapport ont été entièrement négatives. On peut aussi citer le cas du projet *TAUM-AVIATION* développé à l'Université de Montréal qui, dès qu'il s'est lancé dans un effort d'évaluation à large échelle en entreprise, les résultats ont été utilisés pour arrêter le projet.

Donc, il y a un consensus sur le fait qu'il n'existe pas de méthodes d'évaluation généralement reconnues qui permettent d'évaluer d'une manière abstraite la qualité d'un système, ni de comparer d'une manière fiable différents systèmes entre eux. Par

contre, il est accepté que des métriques puissent être développées pour comparer plusieurs versions d'un même système afin de faciliter le développement. L'idée a été soulevé depuis longtemps par Margaret King (1996) ainsi que Flickinger *et al.* (1987). Dans le contexte de l'évaluation du progrès et du diagnostique, ils proposent que même si on ne peut pas développer une méthode pour évaluer les systèmes de traitement de la langue en général, il doit être possible et utile de développer une méthodologie pour une application particulière dans un domaine spécifique.

Plus récemment Srinivas *et al.* (1996) indique que l'évaluation de la performance d'un parseur a deux usages. Premièrement il quantifie la performance d'une grammaire utilisée par un parseur, détermine ses faiblesses et donne une direction pour un développement productif de la grammaire. Un autre objectif de l'évaluation est la comparaison de la performance de différents parseurs. Cela requiert une métrique d'évaluation qui ne prend pas en compte les différences de représentations entre les sorties des différents parseurs.

Cette dernière citation touche directement notre travail "Donner une direction pour un développement productif de la grammaire", c'est précisément ce que notre méthode d'évaluation vise à faire. Cette citation révèle aussi pourquoi il n'y a pas d'antécédent à ce que nous faisons parce que toujours selon Srinivas *et al.* (1996), un autre objectif de l'évaluation est la comparaison de la performance de différents parseurs. Notre méthode d'évaluation ne répond pas à ce deuxième critère et nous ne croyons pas qu'il soit possible de bien réussir ces deux objectifs avec une seule méthode d'évaluation.

Différents auteurs ont classifiés et regroupés les mesures selon plusieurs catégories. Nous présentons maintenant les trois classifications les plus rencontrées dans la littérature : Black box et Glass box; Intrinsic and Extrinsic ainsi que Gold standard. Nous présentons l'évaluation *Feature-based*, qui est une classe à part, parce que l'on emploit le terme parfois pour désigner un type d'évaluation, parfois pour désigner la norme ISO qui a conduit à la création d'une méthode d'évaluation certifiée. Puis nous

présentons la norme d'évaluation la plus connue, soit Parseval, suivi de la présentation de quelques mesures empiriques dont [incr tsdb()]. En terminant, nous soulignons le seul article que nous avons trouvé qui s'attarde au problème de l'évaluation des grammaires informatiques.

2.1 Black box et Glass box

La classification des sytèmes d'évaluation, entre *Glass box* et *Black Box* (voir Palmer et Finin (1990)) est une des premières à avoir été présentés. Lors d'une évaluation par *Black box*, le système est fermé à l'évaluateur. Il n'a accès qu'aux paires entrés/sorties produites par le système. Par opposition dans les évaluations *Glass box*, l'évaluateur a accès aux différentes composantes du système et il peut les évaluer séparément. La classification est semblable mais pas identiques aux évaluations *White box* et *Black box* utilisés en informatique. *Glass box* réfère plutôt aux tests des sous-composantes tandis que *White box* réfère à l'accès au code source pour déterminer les tests à effectuer.

2.2 Intrinsic and Extrinsic measures

Dans un rapport interne sur l'évaluation des systèmes de traitement de la langue, Galliers et Jones (1993), qui fut ensuite repris dans un livre Jones et Galliers (1996), on arrive à des conclusions peu flatteuses sur les systèmes d'évaluation existants : la valeur des évaluations est très limitée. On y postule aussi qu'il est nécessaire d'évaluer un système à plus d'un niveau.

Le niveau *Intrinsic* évalue le système d'une manière indépendante tandis que le niveau *Extrinsic* va évaluer les effets du système étudié sur un système plus vaste. Le niveau *Intrinsic* évalue le système en vase clos. C'est une évaluation en laboratoire où tous

les paramètres sont contrôlés. On ne fait pas de distinctions si l'évaluateur a une connaissance du fonctionnement interne du système ou non. Ce qui importe, c'est que le système soit évalué d'une manière isolée des autres composantes avec lequel il peut interagir.

Le niveau *Extrinsic* évalue le système en l'intégrant dans un plus grand système. Le système ne sera pas évalué directement. C'est plutôt l'effet qu'il aura sur le plus grand système qui sera pris en compte. Par exemple on peut évaluer une grammaire en l'insérant dans un système de traduction automatique et en observant les résultats obtenus.

2.3 Gold standard

Les méthodes d'évaluations basés sur des *Gold standard* se sont beaucoup développées entre autre grâce aux succès obtenus dans le domaine de la recherche d'informations (voir Goodman (1996) et les conférences LREC). Ces méthodes sont basées sur l'établissement d'un corpus d'évaluation annoté avec les résultats attendus pour chacun des cas dans le corpus, d'où le nom de *Gold standard*, ensuite un *baseline* est établi et on calcule la précision et le rappel. La difficulté dans l'utilisation de cette méthode pour l'évaluation des grammaires est le temps nécessaire pour développer un corpus annoté et l'obtention d'un consensus sur le résultat attendu pour l'analyse d'une phrase.

Il faut souligner l'utilisation intensive des *gold standard* dans le domaine particulier du *corpus based parsing*, qui consiste à générer une grammaire par des méthodes statistiques et probabilistes à partir d'un grand ensemble de textes déjà analysés (voir Goodman (1996)).

2.4 Feature-based

L'évaluation *Feature-based* est la seule méthode d'évaluation pour les outils de traitement de la langue basée sur une norme ISO EAGLES (1996). C'est une évaluation du style Consumer Report (*Protégez-vous* au É.U.), où on énumère une liste de fonctionnalités du système et où on décrit comment le système y répond. Des systèmes peuvent être comparés entre eux selon diverses fonctionnalités mais il n'y a pas de notes globales qui sont déterminés. À notre connaissance, ce système n'a jamais été utilisé.

2.5 Parseval

Parseval (Harrison *et al.* (1991)) est le *Gold standard* le plus connu pour l'évaluation des grammaires. C'est un sous-ensemble annoté d'un corpus équilibré. Il contient une méthode de mesure qui indique le nombre de fois qu'un parseur se trompe dans l'analyse d'une phrase par rapport au corpus de référence. Les avantages de Parseval est qu'il utilise un corpus de référence bien connu dont les limitations sont bien connues aussi. Et la métrique qu'il propose est facile à calculer et à comprendre.

Parseval a aussi beaucoup de détracteurs. On lui reproche d'être biaisé envers un type particulier de représentation grammaticale et de défavoriser les analyseurs plus précis qui font plus souvent des erreurs mineures par rapport aux analyseurs qui font une analyse moins précises, mais qui ne se prononcent pas sur plusieurs rattachements entre les mots. Pour paraphraser Dekang Lin (1998), qui résume la situation : même si Parseval a été très utile, plusieurs chercheurs sont insatisfaits des résultats. Leurs propres jugements intuitifs de ce qu'est une bonne analyse est souvent à l'opposé des résultats obtenus par Parseval. On cite souvent : *Parseval est facile à calculer, mais il y a peu de vertus dans une mesure facile à calculer si elle calcule la mauvaise*

chose. Parseval calcule la conformité avec l'analyse de référence mais que se passe-t-il si l'analyse produite est plus fine que celle du corpus de référence ? Bangalore *et al.* (1998) explique *À cause de ces limitations, il n'est pas clair comment le score à cette métrique est relié au succès d'une analyse. Ce n'est pas clair non plus si cette métrique peut être utilisée pour comparer des parseurs avec des degrés de finesses de structures différents puisque le résultat de cette métrique est intimement relié au degré de détails dans les structures du* gold standard.

Quelques améliorations aux méthodes de calculs ont été proposées, dont celles de Lin (1998) et de Srinivas *et al.* (1996), pour rendre l'évaluation possible avec d'autres formalismes linguistiques, mais qui rendent le calcul de la métrique beaucoup moins simple.

2.6 Mesures empiriques

Une autre approche à l'évaluation des grammaires est simplement l'accumulation de statistiques pendant les opérations de parsage, comme le décrit Oepen et Callmeier (2004) dans son article *Measure for Measure.* La plupart des systèmes possèdent des outils qui permettent d'analyser en lot toutes les phrases d'un corpus. Les résultats des analyses ainsi que quelques autres paramètres sont enregistrés dans un *fichier log.* Ce fichier est possiblement analysé pour s'assurer qu'il n'y a pas de régressions. Les paramètres enregistrés sont habituellement : le nombre d'analyses ambiguës, la vitesse d'analyse et la consommation de mémoire.

Le champion dans ce domaine est sans doute le système [incr tsdb()] de Oepen et Callmeier (2004) qui utilise un système emmagasinant plus d'une centaine de paramètres durant l'analyse. La méthode repose sur la comparaison des logs en effectuant le même travail sur un ordinateur différent ou en remplaçant une composante du système par une composante développée par une autre équipe. Plusieurs succès de débogage et

d'optimisation ont été décrits en utilisant ce système, les problèmes étaient souvent causés par des versions de librairies inefficaces sur certains systèmes d'exploitation. Il ne semble pas facile d'utiliser ce système pour détecter des problèmes internes à la grammaire. Pour l'instant il n'est implanté que pour un groupe de grammaires HPSG.

Une des mesures centrales utilisée dans plusieurs systèmes et proposée par Charniak *et al.* (1998) et Roark et Charniak (2000) est le calcul du nombre d'opérations dans l'algorithme de parsage avant d'arriver à une analyse. Ce résultat plus abstrait permettrait de comparer des analyseurs différents implantés dans des langages différents. Moore (2000) de chez Microsoft Research conteste cette méthode dans son article *Time as a Measure of Parsing Efficiency* et propose plutôt d'utiliser des implémentations standardiser des principaux algorithmes de références et d'utiliser de temps requis pour faire une analyse afin de comparer des grammaires entre elles. Des implantations de plusieurs algorithmes en Perl sont d'ailleurs disponibles sur son site web.

2.7 L'évaluation des grammaires informatiques

Un domaine connexe et peu développé lui aussi est l'évaluation des grammaires informatiques. Nous n'avons relevé qu'un seul groupe qui s'y est intéressé : Power et Malloy (2004). Ils décrivent leur approche comme du *Grammar Engineering* soit dans le contexte où des métriques sont utilisées pour estimer l'effort requis pour faire la maintenance d'un logiciel ainsi que pour faciliter l'analyse de l'impact d'un changement. Ils proposent d'étendre le domaine d'applicabilité de ces métriques logicielles afin de couvrir aussi les logiciels qui sont fortement basés sur des grammaires tel que les compilateurs, les éditeurs, les outils de compréhension de programmes et des systèmes embarqués. Ils ont donc adaptés les métriques logicielles les plus utilisés en se basant sur les études théoriques des grammaires de Brauer (1973) et Csuhaj-Varjú;

et Kelemenová(1993). SynQ l'outil qu'ils ont développé a été appliqué sur les grammaires des langages Oberon, C, C++, C# et Java. Malheureusement les métriques ont ne traitent que les grammaires BNF et EBNF, il n'est pas possible d'appliquer directement ces travaux aux grammaires d'unifications.

CHAPITRE 3

L'ANALYSE STATIQUE

3.1 Analyse statique des programmes impératifs

L'analyse statique est souvent décrite comme la technique à la base des optimisations effectuées par les compilateurs, mais elle permet de réaliser encore plus de choses telles que la détection automatique de certains bogues, la détection des clones et la génération de messages d'avertissements. L'introduction classique à l'analyse statique est le chapitre 10 du livre de Aho *et al.* (1986) qui traite de l'optimisation du code. Une référence plus complète (et plus aride) est le livre de Nielson *et al.* (1999).

Ce chapitre présente les notions de bases de l'analyse statique utilisée dans le reste de ce travail. L'analyse statique sera rapidement introduite d'une manière théorique, puis un exemple sera donné. L'exemple effectué sera celui des définitions accessibles (*reaching definitions*) qui permet de déterminer qu'elle est la portée de la définition d'une variable dans un programme. L'exemple est une simplification d'un problème présenté au chapitre 17 de Appel (1997).

L'analyse statique est une technique d'approximation. Car l'analyse des programmes informatiques est un problème indécidable. Il n'existe pas une méthode qui permet de dire si une ligne en particulier dans un programme est atteignable ou non sans exécuter ce programme qui contient potentiellement une boucle infinie avant d'y arriver. Il n'existe pas non plus une méthode qui permet de déterminer exactement quelles sont les valeurs possibles d'une variable à un endroit précis du programme. L'analyse statique va répondre d'une manière partielle à ces questions. À la question *Quel sont les valeurs possibles pour une variable ?* Une analyse statique pourra répondre : voici

un ensemble de valeurs que nous savons possibles pour cette variable, mais notez qu'il pourrait aussi y en avoir d'autres valeurs possibles mais que nous n'avons pas identifiées. Une autre analyse statique répondra : voici un ensemble de valeurs que nous savons impossibles, mais notez qu'il pourrait en avoir d'autres qui sont impossibles mais nous n'avons pas identifiées. Les résultats ainsi présentés sont approximatifs mais pas complètement inutiles. Prenons l'exemple où l'ensemble des valeurs connues impossibles contient toutes les valeurs sauf une, nous pouvons affirmer d'une manière certaine qu'il n'y a au maximum qu'une seule valeur possible pour cette variable. On pourrait, sans risque de se tromper, optimiser cette variable en une constante ou bien générer un avertissement pour signaler l'erreur potentielle. La technique ne permettra pas de trouver tous les cas où l'on peut remplacer une variable par une constante mais elle pourra en identifier une partie.

Les analyses statiques se décrivent formellement en quatre parties : le graphe de flux, le treillis des solutions, les équations de flux et la direction de l'analyse. Nous allons les décrire séparément et à la fin un exemple illustrera comment elles se complètent.

Le graphe de flux représente le programme étudié, les noeuds étant les instructions et les arcs représentant les changements de contrôles entre les instructions. Les arcs représentent surtout le flot de l'information entre les instructions. Les valeurs définies pour les variables vont se propager en suivant les arcs entre les noeuds. La définition précise de ce qui constitue un noeud ou un arc variera selon les problèmes et selon le type d'analyse effectuée.

Le treillis des solutions représente un ordonnancement des différentes solutions possibles. Un ordre partiel suffit, l'ordonnancement étant alors représenté sous forme d'un treillis. La résolution du problème de l'analyse statique se fera par une technique itérative. L'ordonnancement des solutions permet de garantir que si l'on déplace toujours dans la même direction dans l'espace des solutions, il n'y aura pas de cycles et l'algorithme itératif convergera.

Les équations de flux vont préciser quelles valeurs se propagent sur les arcs et comment ils seront fusionnés lorsque plusieurs arcs se rencontrent.

La direction de l'analyse est la dernière composante pour décrire une analyse statique. Dans la plupart des analyses les valeurs vont se propager sur les arcs selon la même direction que le programme s'exécute mais ce n'est pas obligatoire. Certaines analyses s'effectuent dans l'ordre inverse, se sont les valeurs attendues qui se propagent vers les définitions possibles.

3.2 Exemple : définitions disponibles

L'exemple illustré ici est l'analyse des définitions disponibles (*reaching definition*). Elle détermine si un assignement particulier à une variable t (*temporary*) peut directement affecter la valeur de t à un autre endroit dans le programme. Les instructions qui nous intéressent sont les opérations d'affectations de la forme $d : t = a$, où d est le numéro de la ligne où une valeur est affectée à t et dont a peut être une expression ou une constante. L'analyse des définitions disponibles se définit donc ainsi : la définition d à la ligne u atteint l'instruction i dans le programme s'il y a un chemin dans le graphe de flux qui permet de relier i à u sans qu'il y est une autre définition de t sur le chemin.

Nous précisons que toutes les instructions qui ne sont pas des affectations de t ne nous intéressent pas pour cette analyse car elles n'influencent pas le résultat. Les affectations de t ont deux rôles : d'abord elles produisent une nouvelle valeur qui indique une affectation à t, ensuite, elle bloque aussi la propagation de toutes autres définitions de t qui auraient pu être effectuées avant.

La Figure 3.1 montre le programme que nous allons étudier avec l'analyse des définitions disponibles. Il contient les deux variables a, définie aux lignes 1 et 6, et c définie

aux lignes 2, 4 et 7.

```
1 :       a = 5
2 :       c = 1
3 : L1 :  if c > a goto L2
4 :       c = c + c
5 :       goto L1
6 : L2 :  a = c - a
7 :       c = 0
```

FIGURE 3.1 Programme étudié par l'analyse des définitions disponibles

Le graphe de flux indique la direction du contrôle entre les instructions et correspond au flot de la propagation des définitions entre les instructions.

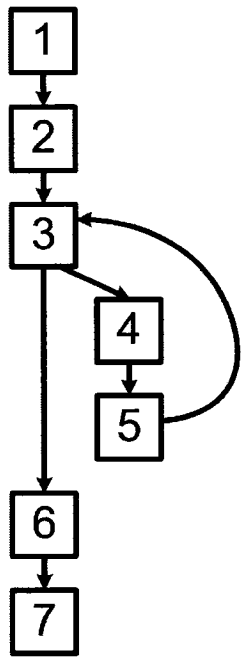

FIGURE 3.2 Graphe de flux

La solution à l'analyse des définitions disponibles s'exprime sous la forme de la liste des définitions qui sont disponibles pour une certaine instruction. Ces solutions sont ordonnées dans un treillis illustré par la Figure 3.3 : à la base du treillis aucune définition n'est accessible à cette instruction, au sommet toutes les définitions sont

accessibles, entre les deux les solutions sont réparties selon l'opération d'inclusion des ensemble.

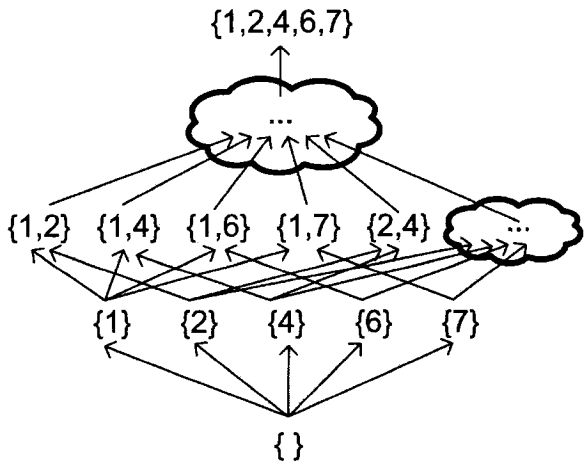

FIGURE 3.3 Extrait du treillis des solutions des définitions disponibles pour une certaine instruction du programme.

Avant de donner les équations, nous définissons les fonctions suivantes. $gen[x]$ retourne l'ensemble des définitions générées à la ligne x. C'est à dire que $gen[x] = x$ si la ligne x est une définition et $gen[x] = \varnothing$ si la ligne n'en est pas une. $kill[x]$ retourne l'ensemble des définitions qui ne seront plus disponibles après cette ligne c'est-à-dire l'ensemble de toutes les définitions de cette variable sauf celle définie à la ligne x. Pour notre programme $kill[4]$ sera $2, 7$ parce que l'affectation de c qui est définie aux lignes 2, 4 et 7 bloquera toutes les définitions sauf celle de 4. $pred[x]$ réfère au graphe de flux et retourne l'ensemble des noeuds qui précèdent immédiatement le noeud x. Le résultat de ces fonctions peut être pré-calculé pour tout le programme afin d'accélérer la solution itérative, comme c'est illustré dans le Tableau 3.1.

Voici les équations pour la progagation des données, elles donnent les valeurs à l'entrée et à la sortie d'un noeud n :

TABLEAU 3.1 Fonctions définies sur le programme étudié

n	gen[x]	kill[x]	pred[x]
1	{1}	{6 }	∅
2	{2}	{4, 7}	{1}
3	∅	∅	{2, 5}
4	{4}	{2, 7}	{3}
5	∅	∅	{4}
6	{6}	{1}	{3}
7	{7}	{2, 4}	{6}

$$in[n] = \bigcup_{p \in pred[n]} out[p] \tag{3.1}$$

$$out[n] = gen[n] \bigcup (in[n] - kill[n]) \tag{3.2}$$

En utilisant les fonctions déjà définies, on peut calculer itérativement quel est l'ensemble des valeurs qui atteignent l'entrée (ou la sortie) d'un noeud n en utilisant par exemple l'algorithme classique de Aho *et al.* (1986) tel que reproduit à la Figure 3.4. Le résultat obtenu est illustré dans les trois sections du tableau 3.2. La première section donne les numéros de ligne du programme n. La seconde section rappelle les valeurs de *gen* et *kill* pour simplifier la lecture du tableau. La troisième section contient trois colonnes qui correspondent à trois itérations de l'algorithme itératif. La troisième itération sert seulement à découvrir que rien n'a changé depuis l'itération précédente et que l'algorithme peut s'arrêter.

On peut interpréter les résultats obtenus de plusieurs manières. Si l'on veut faire de la propagation de constantes, on va remarquer qu'une seule définition de a (celle de 1) peut atteindre l'instruction 3 et que l'on peut remplacer $c > a$ par $c > 5$.

```
1:  for each block B do out[B] := gen[B];
2:  change := true;
3:  while change do begin
4:      change := false;
5:      for each block B do begin
6:          in[B] := UNION( out[P] ) for each P predecesor of B
7:          oldout := out[B];
8:          out[B] :=  gen[B] U (in[B] - kill[B]);
9:          if out[B] != oldout then change := true
        end
    end
```

FIGURE 3.4 Algorithme pour calculer les *in* et les *out*. Tiré de Aho et al. (1986) à la Figure 10.26.

TABLEAU 3.2 Solution itérative

n	gen[n]	kill[n]	in[n]	out[n]	in[n]	out[n]	in[n]	out[n]
1	1	6		1		1		1
2	2	4,7	1	1,2	1	1,2	1	1,2
3			1,2	1,2	1,2,4,	1,2,4	1,2,4,	1,2,4
4	4	2,7	1,2	1,4	1,2,4,	1,4	1,2,4,	1,4
5			1,4	1,4	1,4	1,4	1,4	1,4
6	6	1	1,2	2,6	1,2,4	2,4,6	1,2,4	2,4,6
7	7	2,4	2,6	6,7	2,4,6	6, 7	2,4,6	6, 7

3.3 Analyse statique des programmes logiques

Nous n'avons pas trouvé de travaux qui décrivent l'application de l'analyse statique sur une grammaire. Ce que nous avons trouvé qui s'en rapproche le plus est l'analyse statique sur des langages logiques comme Prolog. La mention est importante car les grammaires sont aussi des langages logiques. On peut donner comme preuve que les grammaires peuvent directement se traduire et s'exécuter en Prolog.

La principale motivation à réaliser l'analyse statique des programmes Prolog est d'op-

timiser l'utilisation de la mémoire et du ramasse-miette (*garbage collection*). C'est ce qui est effectué par Mulkers *et al.* (1994). Il utilise les travaux de Cousot et Cousot (1992) et Debray (1992) qui explorent différentes approches à cette analyse statique d'une manière qui soit efficace en temps et en consommation mémoire.

Les problèmes mis en oeuvre pour l'optimisation du ramasse-miette est beaucoup plus complexe que l'exemple simple des définitions disponibles. Il nécessite de prendre en compte, entre autres, la notion d'alias (pointeur), ce qui change la dynamique de l'analyse. Comme les grammaires d'unifications n'utilisent pas cette possibilité de Prolog, nous pouvons utiliser un modèle plus simple qui ressemble plus au modèle de la section précédente.

CHAPITRE 4

MODÈLE DE PROPAGATION DES VALEURS POSSIBLES

Ce chapitre présente notre adaptation d'une technique de l'analyse statique pour déterminer les valeurs possibles des variables dans une grammaire d'unification. Le problème peut se résume à ceci : pour n'importe quelle règle r de la grammaire et pour n'importe quel attribut a, déterminer l'ensemble V des valeurs que a peut prendre. L'analyse sera décrite selon les quatre aspects formels d'une analyse statique, soit la construction du graphe de flux, les équations de propagations des valeurs, le treillis des solutions possibles et la direction de l'analyse.

Nous présentons ici deux solutions à ce problème. La première solution est une adaptation directe d'une analyse pour un langage impératif comme vu dans le chapitre précédent. La seconde solution, qui est toujours en développement, raffine cette analyse en traitant le cas des attributs récursifs et en considérant aussi l'origine des différentes valeurs. Nous avons utilisé les résultats de la première analyse pour valider ceux obtenus par la seconde. Nous avons inclus un exemple détaillé du fonctionnement avec la seconde analyse afin de simplifier la compréhension.

4.1 Premier modèle

Définissons d'abord les grammaires d'unification pour simplifier les équations qui vont suivre. Nous ajoutons à la définition classique des grammaires l'ensemble $FEAT$ qui va représenter les attributs utilisés dans les structures de la grammaire. Donc, soit G une grammaire d'unification définie comme suit :

$$G = (N, T, RULES, S, FEAT)$$

$$propagSymbol : RULES \rightarrow N$$

$$propagSet : RULES \rightarrow$$
$$\rightarrow FEAT \times \{VARS \cup ATOMS\} \tag{4.1}$$

$$unifySets : RULES \rightarrow$$
$$\rightarrow (FEAT \times \{VARS \cup ATOMS\})^n, \ (n \in \mathcal{N}^+)$$

$$vars : RULES \rightarrow VARS$$

où N est l'ensemble des non-terminaux, T est l'ensemble des terminaux (les items lexicaux ou jetons), $RULES$ est l'ensemble des règles de production utilisées, S est le symbole de départ, et $FEAT$ est l'ensemble des attributs propagés par les règles de la grammaire.

Quelques fonctions sont définies sur les règles. $propagSymbol$ retourne le non-terminal de la partie de gauche d'une règle, $propagSet$ retourne l'ensemble des attributs et leurs valeurs propagées par une règle, $unifySets$ retourne le n-tuple formé par les ensemble des attributs et leurs valeurs ou variables utilisés par la partie de droite d'une règle et $vars$ retourne l'espace de nom de la variable dans une règle.

4.1.1 Graphe de flux

Définissons $DG = (V_{DG}, E_{DG})$ comme le graphe de flux (*Derivation Graph*) tel que :

$$(V_{DG} = N \cup T)$$

$$\begin{gathered}(v_1,\ v_2 \in V_{DG}), (v_1, v_2) \in E_{DG} \longleftrightarrow \\ (\exists r_1,\ r_2 \in RULES \mid \\ (v_1 \in LHS(r_1)) \wedge\ (v_2 = RHS(r_2)))\end{gathered} \tag{4.2}$$

4.1.2 Treillis

Un treillis des solutions pour le problème de l'analyse de flux peut être construit en considérant l'ordre partiel existant entre les ensembles des valeurs des attributs. $\perp$ représente le noeud du treillis pour lequel tous les attributs n'ont aucune valeur possible. $\top$ représente le noeud pour lequel tous les attributs ont toutes les valeurs possibles dans leurs domaines. Un noeud S_i dans le treillis représente une configuration des valeurs possibles de tous les attributs dans la grammaire.

Supposons qu'il y a n attributs distincts nommés de a_1 à a_n dans la grammaire G. Formellement, un noeud S_i est caractérisé par l'information de flux qui lui est associé comme suit :

$$S_i = (V_{i,1}, \cdots, V_{i,k}, \cdots, V_{i,n}) \tag{4.3}$$

où $V_{i,k} \subseteq \mathcal{P}(ATOMS)$ est l'ensemble des valeurs associées à l'attribut a_k.

Le sommet et la base du treillis sont respectivement :

$$\begin{aligned}\top &= (V_\top, \cdots, V_\top, \cdots, V_\top) \\ \perp &= (V_\perp, \cdots, V_\perp, \cdots, V_\perp)\end{aligned} \tag{4.4}$$

où $V_\top$ est un symbole spécial indiquant que l'attribut a_k est associé à un ensemble de valeurs qui est égal à l'ensemble complet des valeurs et $V_\bot$ est un symbole spécial indiquant que l'attribut a_k est associé à un ensemble vide de valeurs.

Pour n'importe quel noeud S_i dans le treillis, l'ordre partiel se définit ainsi :

$$\begin{aligned} \bot &\preceq S_i \\ S_i &\preceq \top \end{aligned} \tag{4.5}$$

Pour n'importe quelle paire de noeuds S_i et S_j dans le treillis qui ne sont ni $\top$ ni $\bot$, l'ordre partiel entre les noeuds est défini par l'inclusion ensembliste entre les ensembles des valeurs correspondant à tous les attributs de la grammaire.

$$S_i \preceq S_j \longleftrightarrow V_{i,k} \subseteq V_{j,k}, 1 \leq k \leq n \tag{4.6}$$

4.1.3 Équations de flux

Les équations de propagation des attributs pour une règle de grammaire r sont décrites en terme de l'information de flux entrant (in) et sortant (out) de la règle r. Les ensembles $IN(r)$ et $OUT(r)$ décrivent cette information et correspondent à des noeuds dans le treillis des solutions.

Au départ, l'analyse commence avec des ensembles vides pour tous les attributs, ce qui est

$$\forall r \in RULES, IN(r) = OUT(r) = (V_\bot, \cdots, V_\bot, \cdots, V_\bot) = (\emptyset, \cdots, \emptyset, \cdots, \emptyset). \tag{4.7}$$

Définissons l'information de flux entrant dans la règle r comme :

$$IN(r) = (V_{i,1}, \cdots, V_{i,k}, \cdots, V_{i,n}) \tag{4.8}$$

et l'information de flux sortant de r comme :

$$OUT(r) = (V_{j,1}, \cdots, V_{j,k}, \cdots, V_{j,n}) \tag{4.9}$$

Les éléments de $OUT(r)$ peuvent être calculés comme suit :

$$V_{j,k} = \begin{cases} V_{\perp} & \text{if } \nexists\ (a_k,\ v) \in propagSet(r) \\ \{v\} & \text{if } (a_k,\ v) \in propagSet(r) \wedge \\ & \quad (v \in ATOMS) \\ \bigcup\limits_{V_h \in IN(r)\ \mid} V_h & \text{if } (a_k,\ x) \in propagSet(r) \wedge \\ \quad (\exists\ unifySet_i \in unifySets(r) \mid & \quad (x \in VARS(r)) \\ \quad (a_h,\ x) \in\ unifySet_i) & \end{cases} \tag{4.10}$$

Lorsque plusieurs arcs du graphe de flux de la grammaire fusionne dans une même règle, il est nécessaire du fusionner aussi l'information de flux venant de r_i et r_j, par exemple, pour obtenir l'information de flux atteignant r_m. Supposons que $OUT(r_i) = (V_{i,1}, \cdots, V_{i,k}, \cdots, V_{i,n})$ et $OUT(r_j) = (V_{j,1}, \cdots, V_{j,k}, \cdots, V_{j,n})$.

L'information fusionnée est :

$$IN(r_m) = OUT(r_i) \bowtie OUT(r_j) = (V_{m,1}, \cdots, V_{m,k}, \cdots, V_{m,n}) \tag{4.11}$$

où $\bowtie$ est l'opérateur de fusion et

$$\forall k, (1 \leq k \leq n), V_{m,k} = V_{i,k} \cup V_{j,k} \tag{4.12}$$

Puisque les équations présentés préservent l'ordre partiel défini par les équations 4.5 et 4.6, la solution par point-fixe et garantie de converger.

4.1.4 Direction de l'analyse

La direction de l'analyse est avant puisque nous calculons l'ensemble des valeurs d'un noeud à partir des valeurs des noeuds précédents dans le graphe de flux.

4.2 Second modèle

Ce second modèle reprend toutes les fonctionnalités du premier et ajoute certains raffinements, soient le traitement des attributs récursif et l'origine des valeurs propagées en plus de modéliser plus finement le comportement de l'unification à l'intérieur d'une règle. Ces raffinements sont nécessaires afin d'avoir des résultats plus précis.

Nous définissons plus précisément la grammaire G comme ceci :

$$G = (N, FEAT, VARS, FSTR, ATOMS, RULES, S)$$

$$\begin{gathered} FSTR \ : \mathcal{P}\{attval \mid attval \in FEAT \times (ATOMS \cup VARS \cup FSTR)\} \\ propagSet : RULES \rightarrow FSTR \\ unifySets : RULES \rightarrow (FSTR)^n, \ (n \in \mathcal{N}^+) \end{gathered} \tag{4.13}$$

où N (non-terminal) est l'ensemble des symboles syntaxique, $FEAT$ (feature) est l'ensemble des attributs, $VARS$ est l'ensemble des variables utilisés avec l'unification, $FSTR$ (feature structures) est l'ensemble des structures d'attributs, $ATOMS$ l'ensemble des valeurs atomiques présents dans la grammaire, $RULES$ est l'ensemble des règles de grammaires et S (start) est le symbole de départ. Remarquez que la valeur d'un un attribut $FEAT$ peut être une structure d'attribut $FSTR$, ceci permettant les structures d'attributs imbriqués.

Quelques fonctions sont définies sur les règles. La fonction $propagSet$ retourne la structure d'attribut de la partie de gauche de la règle. La fonction $unifySet$ retourne le n-tuple des structures présentes dans la partie de droite de la règle.

4.2.1 Graphe de flux

À partir de ce formalisme grammatical, nous construisons un graphe de flux qui représentera essentiellement comment les valeurs des attributs peuvent être propagés pendant le processus de parsage. Un noeud dans ce graphe correspond à une structure d'attribut dans une règle de grammaire. Trois types d'arcs sont définis. Un arc $PROPAG$ associe la structure d'attribut de la partie de gauche d'une règle avec toutes les structures de la partie de droite de cette même règle. Un arc $UNIF$ connecte la partie de gauche d'une règle à chacune des parties de droite des autres règles (ou de la même règle) avec qui l'unification est possible. Simplement, les arcs $PROPAG$ et $UNIF$ représentent le flux de la propagation des valeurs obtenues par l'unification des structures d'attributs.

Pour prendre en compte le fait que certaines variables peuvent apparaître plus d'une fois dans la partie de droite d'une règle, nous définissons aussi l'arc de type $SYNC$.

Règles :

(1) $S \rightarrow NP\begin{bmatrix} \text{AGR} & A \\ \text{PERS} & P \end{bmatrix}, VP\begin{bmatrix} \text{AGR} & A \\ \text{PERS} & P \end{bmatrix}$

(2) $NP\begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{NUM} & N \\ \text{GEN} & G \end{bmatrix} \\ \text{PERS} & 3 \end{bmatrix} \rightarrow DET\begin{bmatrix} \text{NUM} & N \\ \text{GEN} & G \end{bmatrix}, N\begin{bmatrix} \text{NUM} & N \\ \text{GEN} & G \end{bmatrix}$

(3) $NP\begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{NUM} & N \\ \text{GEN} & G \end{bmatrix} \\ \text{PERS} & P \end{bmatrix} \rightarrow PRON\begin{bmatrix} \text{NUM} & N \\ \text{GEN} & G \\ \text{PERS} & P \end{bmatrix}$

(4) $VP\begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{NUM} & N \end{bmatrix} \\ \text{PERS} & P \end{bmatrix} \rightarrow V\begin{bmatrix} \text{NUM} & N \\ \text{PERS} & P \\ \text{VAL} & \text{intr} \end{bmatrix}$

(5) $VP\begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{NUM} & N \end{bmatrix} \\ \text{PERS} & P \end{bmatrix} \rightarrow V\begin{bmatrix} \text{NUM} & N \\ \text{PERS} & P \\ \text{VAL} & \text{trans} \end{bmatrix}, NP$

Lexique :

(6) $DET\begin{bmatrix} \text{NUM} & \text{sing} \\ \text{GEN} & \text{masc} \end{bmatrix} \rightarrow$ le

(7) $DET\begin{bmatrix} \text{NUM} & \text{plur} \\ \text{GEN} & \text{masc} \end{bmatrix} \rightarrow$ les

(8) $N\begin{bmatrix} \text{NUM} & \text{sing} \\ \text{GEN} & \text{masc} \end{bmatrix} \rightarrow$ garçon

(9) $N\begin{bmatrix} \text{NUM} & \text{plur} \\ \text{GEN} & \text{masc} \end{bmatrix} \rightarrow$ garçons

(10) $PRON\begin{bmatrix} \text{NUM} & \text{sing} \\ \text{PERS} & 1 \end{bmatrix} \rightarrow$ je

(11) $V\begin{bmatrix} \text{NUM} & \text{sing} \\ \text{PERS} & 3 \\ \text{VAL} & \text{intr} \end{bmatrix} \rightarrow$ dors

FIGURE 4.1 Exemple d'une grammaire pour le français

Il relie deux structures d'attributs présent dans la partie de droite d'une même règle et qui partagent au moins une variable.

La Figure 4.2 montre le graphe de flux de la petite grammaire du portugais de la Figure 4.1. Les détails de cette figure sont expliqués après la définition formelle de ce graphe. Notez que les arcs $PROGAG$ et $UNIF$ sont représentés par une ligne continues alors que les arcs $SYNC$ sont représentés par des lignes discontinues.

Pour rendre la propagation de l'information sur les attributs sensibles à l'unification, les noeuds ont étés définis de deux type différents. Un noeud est du type LHS (left hand side) s'il représente une structure d'attribut associé à la partie de gauche d'une règle de grammaire. De la même manière, un noeud est du type RHS s'il correspond à une structure d'attribut de la partie de droite d'une règle. L'information de flux d'un noeud LHS peut être propagé à tous les noeuds RHS qui sont unifiable avec lui. L'information de flux à partir d'un noeud RHS peut être seulement propagé au noeud LHS qui lui correspond dans la même règle.

Définissons le graphe de flux (derivation graphe) $DG = (V_{DG}, E_{DG})$, deux fonctions *ntype* et *etype* qui retournent le type d'un noeud et le type d'un arc (respectivement) dans le graphe, la fonction *rule* qui associe des noeuds avec leur règle correspondante et la fonction *order* qui associe des noeuds RHS avec leurs postions dans la partie de droite d'une règle, comme suit :

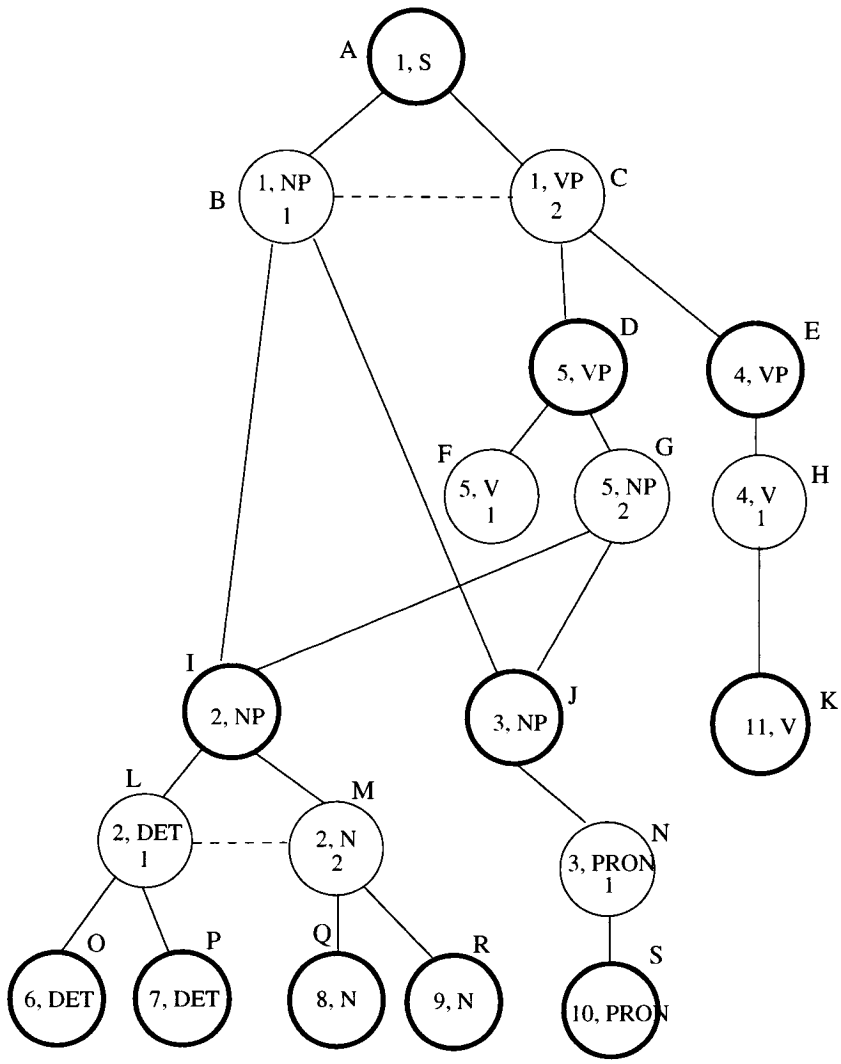

FIGURE 4.2 Example d'un graphe de flux

$$ntype : V_{DG} \rightarrow \{LHS, RHS\}$$

$$etype : E_{DG} \rightarrow \{PROPAG, UNIF, SYNC\} \tag{4.14}$$

$$rule : V_{DG} \rightarrow RULES$$

$$order : V_{DG} \rightarrow \mathcal{N}$$

Les fonctions *ntype*, *etype* et *rule* sont des fonctions totales puisqu'elles sont définies pour chacun des noeuds dans le graphe. La fonction *order* est partielle parce qu'elle est définie seulement pour les noeuds de type LHS. Le graphe est construit comme suit :

- Chaque règle dans la grammaire correspond à un noeud LHS dans le graphe.
- Pour chaque structure d'attribut dans la partie de gauche d'une règle, il y a un noeud unique n de type RHS tel que $order(n)$ correspond à sa position dans la partie de gauche de la règle.
- Si et seulement si la partie de gauche d'une règle est unifiable avec la partie de droite d'une autre règle, il y a un arc $UNIF$ qui relie ces deux noeuds.
- Pour chaque groupe de deux noeuds dans le graphe dont l'un correspond à la structure d'attribut de la partie de gauche d'une règle et l'autre à une structure d'attribut de la partie de droite de cette même règle, il y a un arc $PROPAG$ qui les relies.
- Pour chaque groupe de deux structures d'attribut dans la même partie de droite d'une règle qui partagent une même variable, il y a un arc $SYNC$ qui joint les deux noeuds. Notez que chaque noeud RHS possède un arc $SYNC$ qui le relie à lui-même.

Dans le graphe montré à la Figure 4.2, les noeuds en gras sont utilisés pour distingués les noeuds LHS des noeuds RHS. Dans chacun des noeuds, le numéro de la règle est indiqué avec le symbole syntaxique associé à ce noeud. Pour les noeuds RHS, l'ordre numérique est aussi indiqué. Notez qu'une règle correspondant à une entrée lexicale est représentée par un seul noeud dans le graphe. Pour simplifier la description de l'analyse de flux, une lettre majuscule est accolé à chacun des noeuds pour l'identifier uniquement.

Une fonction additionnelle $fSet$ est définie sur l'ensemble des noeuds pour simplifier la description, à venir, de l'analyse de flux. La fonction $fSet$ retourne la structure d'attribut associé à un noeud dans le graphe et est définie comme suit :

$$fSet : V_{DG} \rightarrow FSTR$$

$$fSet(v) = \begin{cases} propagSet(rule(v)) & \text{si } type(v) = LHS \\ [unifySets(rule(v))]_{order(v)} & \text{si } type(v) = RHS \end{cases} \tag{4.15}$$

où $[t]_i$ retourne le i^{ieme} élément du n-tuple t.

4.2.2 Équation de flux

L'analyse de flux approxime d'une manière statique le comportement dynamique du processus l'unification et de la dérivation en calculant la solution itérative du point fixe des équations de flux.

Essentiellement, c'est un processus itératif qui identifie toutes les valeurs possibles pour toutes les variables. A chacune des itérations, les ensembles sont ajustés tel que chaque variable propage ses valeurs aux autres variables. Le processus s'arrête lorsque

nous arrivons à un état où il n'y a plus de changements dans aucun des ensemble des variables.

L'analyse de flux, que nous allons proposer, est basée sur les valeurs possibles de tous les attributs dans la grammaire. Avant de la présenter, nous devons donner une définition moins restrictive d'un attribut. Nous allons étendre notre définition pour inclure non seulement les attributs simple mais aussi les chemins qui conduisent à une valeur dans une structure imbriquée. Par exemple, le chemin AGR :NUM dans la seconde règle de notre grammaire exemple sera considéré comme un attribut. Nous définissons donc l'ensemble $FEAT^+$ comme suit :

$$FEAT^+ \ : \ FEAT \cup \{f : g \mid f \in FEAT \wedge g \in FEAT^+\} \tag{4.16}$$

En utilisant les chemins d'attributs, nous pouvons éviter les complications dues aux structures imbriquées. Par contre cela requiert de réécrire la grammaire en remplaçant les structures imbriquées par leurs représentations sous forme de chemins d'attributs. Par exemple, les règles (1) et (2) de notre grammaire sont réécrite de la manière illustré à la Figure 4.3. À partir de maintenant, nous allons utiliser le même terme *attribut* pour référer à un attribut simple ou à un chemin dans un attribut imbriqué.

$$\text{(1)} \quad \text{S} \rightarrow \text{NP}\begin{bmatrix} \text{AGR :NUM} & \text{A1} \\ \text{AGR :GEN} & \text{A2} \\ \text{PERS} & \text{P} \end{bmatrix}, \text{VP}\begin{bmatrix} \text{AGR :NUM} & \text{A1} \\ \text{AGR :GEN} & \text{A2} \\ \text{PERS} & \text{P} \end{bmatrix}$$

$$\text{(2)} \quad \text{NP}\begin{bmatrix} \text{AGR :NUM} & \text{N} \\ \text{AGR :GEN} & \text{G} \\ \text{PERS} & 3 \end{bmatrix} \rightarrow \text{DET}\begin{bmatrix} \text{NUM} & \text{N} \\ \text{GEN} & \text{G} \end{bmatrix}, \text{N}\begin{bmatrix} \text{NUM} & \text{N} \\ \text{GEN} & \text{G} \end{bmatrix}$$

FIGURE 4.3 Réécriture des régles (1) et (2) sans structures imbriquées

Le problème de l'analyse de flux peut être défini comme suit. Soit *def-origin* un triplet (b, w, v), où $b \in FEAT^+$, $w \in V_{DG}$ and $ntype(w) = LHS$, $v \in (ATOMS \cup \{ANY\})$, tel que le chemin b existe dans la structure $fSet(w)$ et que sa valeur est v. En plus des valeurs possibles habituelles qui sont : une valeur atomique ou une structure d'attribut, nous définissons la valeur ANY qui représente la valeur d'un attribut absent. Selon la définition de l'unification, lorsqu'une structure d'attribut contient un attribut et qu'elle est unifiée avec une autre structure qui ne contient pas cet attribut, la paire attribut/valeur de la première est copiée dans la seconde. Ainsi la valeur d'un attribut absent peut être n'importe quoi (ANY) dépendamment de ce avec quoi il est unifié. Nous allons voir plus tard que la valeur ANY est très utile pour détecter des erreurs dans la grammaire. En particulier, elle peut révéler l'omission non désirée d'un attribut dans la partie de gauche d'une règle ce qui permet l'activation non voulue de cette règle dans des circonstances non prévues.

En des termes simples, def-origin spécifie la partie de gauche d'une règle dans laquelle un attribut est défini à une valeur, c'est à dire que la valeur de l'attribut n'est pas une variable. Pour chacun des noeuds w dans le graphe et pour chacun des attributs $a \in FEAT^+$, nous pouvons déterminer l'ensemble des def-origins. Donc pour chacun des attributs dans une règle ayant comme valeur une variable, nous pouvons déterminer l'origine de n'importe quelle valeur qui est propagé jusqu'à cet attribut.

Nous avons maintenant les éléments requis pour définir les équations de l'analyse du flux dans une grammaire. Soit $a_1, \ldots a_n$ l'ensemble des attributs distincts qui existe dans la grammaire G. Pour n'importe quelle itération de l'algorithme et pour n'importe quel attribut dans une structure d'attribut d'une règle de grammaire, nous pouvons identifier un ensemble de def-origins pour cet attribut. Nous définissons un état S_i qui représente simplement l'ensemble des def-origins pour tous les attributs à cette itération i. Formellement, un noeud S_i est défini comme suit :

$$S_i = (V_{i,1}, \cdots, V_{i,k}, \cdots, V_{i,n}) \tag{4.17}$$

où $V_{i,k} \subseteq \mathcal{P}(FEAT^+ \times V_{DG} \times (ATOMS \cup \{ANY\}))$ est l'ensemble des def-origins associés avec l'attribut a_k.

Les équations pour calculer les def-origins pour n'importe quelle règle r sont décrites selon l'information de flux entrant ou sortant d'un noeud $w \in V_{DG}$. Nommons $OUT(w)$ la configuration S_i qui représente les def-origins possibles pour chaque attribut $f \in fSet(w)$. Nous allons noter $OUT(a_m, w)$ pour indiquer l'ensemble associé à l'attribut a_m dans la configuration particulière $OUT(w)$.

Initialement, pour chacun des noeuds dans le graphe, l'analyse de flux commence avec un ensemble vide pour chacun des ensembles de tous les attributs, ce qui est $\forall w \in V_{DG}, OUT(w) = (\emptyset, \cdots, \emptyset, \cdots, \emptyset)$. L'information de flux sortant de w est $OUT(w) = (V_1, \cdots, V_k, \cdots, V_n)$. Les éléments de $OUT(w)$ losrque $type(w) = LHS$ peut être calculé comme suit :

$$V_k = \begin{cases} \{(w,\ a_k,\ ANY)\} & \text{if } \nexists\ (a_k,\ v) \in fSet(w) \\ \{(w,\ a_k,\ v)\} & \text{if } (a_k,\ v) \in fSet(w)\ \wedge v \in ATOMS \\ pSet(x,\ w) & \text{if} (a_k, x) \in fSet(w) \wedge (x \in VARS) \end{cases} \tag{4.18}$$

où $pSet(x, w) = OUT(a_j, z)$ pour certains $(z, w) \in E_{DG}$ tel que $etype((z, w)) = PROPAG \wedge (a_j, x) \in fSet(z)$.

V_k représente l'ensemble des def-origins de l'attribut a_k propagés par le noeud courant w. Si a_k n'appartient pas à la partie de gauche de $rule(w)$, c'est à dire que a_k n'appar-

tient pas à l'ensemble des attributs de w, cet attribut peut prendre n'importe quelle valeur puisque l'unification sera toujours réussie. Si la partie de gauche de $rule(w)$ assigne une valeur atomique à un attribut a_k, le def-origin de a_k est alors w lui-même. Sinon, la valeur de l'attribut est une variable et V_k est l'ensemble des def-origins associés à n'importe quelle occurrence de la variable dans la partie de droite de la règle. La raison pour cela, comme nous allons le voir plus tard, est que l'ensemble des def-origins est le même pour toutes les occurrences de la variable dans la partie de droite d'une même règle.

Pour la partie de droite des règles maintenant, les structures d'attributs de la partie de droite peuvent s'unifier avec plusieurs structures d'attributs des parties de gauches. Nous prenons en considération les ensembles de valeurs possibles arrivant à un noeud RHS en calculant l'union des def-origins des noeuds LHS avec lesquelles il est connecté. Définissons $IN(a_k, w)$ comme l'union des ensembles de def-origins associés à l'attribut a_k dans le flux venant des noeuds connectés à w. Formellement :

$$IN(a_k, w) = \bigcup_{\substack{(z,w) \in E_{DG} \\ etype((z,w)) = UNIF}} OUT(a_k, z) \tag{4.19}$$

Soit l'information de flux venant d'un noeud w de type RHS tel $OUT(w) = (V_1, \cdots, V_k, \cdots, V_n)$. Formellement, V_k peut être calculé comme suit :

$$V_k = \begin{cases} IN(a_k, w) & \text{si } \nexists\ (a_k,\ v) \in fSet(w) \\ \{(w,\ a_k,\ v)\} & \text{si } (a_k,\ v) \in fSet(w)\ \wedge v \in ATOMS \\ \bigoplus_{V_i \in syncSet(x,w)} V_i & \text{si}(a_k, x) \in fSet(w) \wedge (x \in VARS) \end{cases} \tag{4.20}$$

où $syncSet(x, w) = \{IN(a_k, z) \mid (z, w) \in E_{DG} \wedge etype((z, w)) = SYNC \wedge (a_k, x) \in fSet(z)\}$. C'est à dire que $syncSet(x, w)$ contient l'ensemble des def-origins pour chaque occurrence de la variable x. Avant de donner la définition de l'opérateur $\oplus$, nous allons l'expliquer intuitivement. Il représente une simulation de l'unification des valeurs de toutes les occurrences d'une variable de la partie de droite d'une règle. Si une valeur peut apparaître dans toutes les occurrences d'une variable, elle sera propagée. Si la valeur spéciale ANY apparaît dans l'ensemble d'une variable, cela signifie que n'importe quelle valeur qui s'unifie avec elle sera propagée. Définissons deux ensembles $A^{+ANY} = \{(a, w, ANY) \mid (a, w, ANY) \in A\}$ et $A^{-ANY} = A/A^{+ANY}$. Formellement, l'opérateur $\oplus$ est définie comme suit :

$$A \oplus B = \begin{cases} A \cup B & \text{si } A^{+ANY} \neq \emptyset \text{ et } B^{+ANY} \neq \emptyset \\ B \cup (A^{-ANY} \sqcap B) & \text{si } A^{+ANY} \neq \emptyset \text{ et } B^{+ANY} = \emptyset \\ A \cup (B^{-ANY} \sqcap A) & \text{si } A^{+ANY} = \emptyset \text{ et } B^{+ANY} \neq \emptyset \\ A \sqcap B & \text{sinon} \end{cases} \tag{4.21}$$

où

$$A \sqcap B = \begin{cases} \{(a_1, w_1, v)\} \cup \{(a_2, w_2, v)\} \cup (A' \sqcap B) & \text{si } A = \{(a_1, w_1, v)\} \cup A' \\ & \text{et } (a_2, w_2, v) \in B \\ \emptyset & \text{sinon} \end{cases} \tag{4.22}$$

Les variables sont traitées différemment dans les noeuds de type RHS. Premièrement, toutes les valeurs possibles pour la variable sont prises en compte à partir des valeurs propagées par les autres règles qui sont unifiables avec la structure d'attribut qui contient cette variable. C'est l'ensemble qui est retourné par la fonction $IN(a_k, w)$. Puis, comme toutes les occurrences de la variable dans une règle doivent avoir les même valeurs, nous ne conservons que les valeurs qui apparaissent dans tous ensembles calculés individuellement (en considérant que ANY représente toutes les valeurs possibles).

L'analyse de flux définie ici effectue seulement une approximation du processus d'unification. Par exemple, si une structure d'attribut contient plus d'une variable, il est assumé que toutes les combinaisons de valeurs sont possibles ce qui n'est pas nécessairement le cas lorsque l'unification est effectuée pendant le parsage d'une phrase. L'identification de toutes les combinaisons valides serait plus couteuse en ressources et il n'est pas clair que c'est nécessaire pour le type d'analyse que nous voulons effectuer. Le modèle plus simple que nous présentons ici donne des résultats utiles.

Retournons à notre grammaire exemple. Un état dans l'analyse de flux va contenir six éléments, un pour chacun des attributs de la grammaire : AGR :NUM, AGR :GEN, VAL, PERS, NUM et GEN. Après quelques itérations, nous obtenons un état stable où il n'y a pas de changements avec la dernière itération voir le tableau **??**. Par exemple,

le noeud S, qui correspond à la partie de gauche de la règle (10), a une seule origine de spécifiée pour l'attribut PERS qui est lui-même. Le champs AGR :NUM du noeud B, qui correspond à la variable A de la première structure d'attribut de la partie de droite de la règle (1), a quatre origine pour la valeur *sing* soit les noeud O, Q, S et K qui correspondent, comme on s'y attend, aux entrées lexicales pour le déterminant, le nom, le pronom et le verbe.

TABLEAU 4.1 Analyse de flux après la dernière itération

Node	AGR :NUM	AGR :GEN	VAL
A	{(AGR :NUM,A,ANY)}	{(AGR :GEN,A,ANY)}	{(VAL,A,ANY)}
B	{(NUM,O,sing), (NUM,Q,sing), (NUM,S,sing), (NUM,K,sing) }	{(GEN,O,masc), GEN,P,masc), GEN,Q,masc), GEN,R,masc), GEN,S,ANY) }	{(VAL,B,ANY)}
C	{(NUM,O,sing), (NUM,Q,sing), (NUM,S,sing), (NUM,K,sing) }	{(GEN,O,masc), GEN,P,masc), GEN,Q,masc), GEN,R,masc), GEN,S,ANY) }	{(VAL,C,ANY)}
D	∅	{(AGR :GEN,D,ANY)}	{(VAL,D,ANY)}
E	{(NUM,K,sing)}	{(AGR :GEN,E,ANY)}	{(VAL,E,ANY)}
F	{(AGR :NUM,F,ANY)}	{(AGR :GEN,F,ANY)}	{(VAL,F,trans)}
G	∅	∅	{(VAL,G,ANY)}
H	{(AGR :NUM,H,ANY)}	{(AGR :GEN,H,ANY)}	{(VAL,H,intr)}
I	{(NUM,O,sing), (NUM,P,plur), (NUM,Q,sing), (NUM,R,plur)}	{(GEN,O,masc), (GEN,P,masc), (GEN,Q,masc), (GEN,R,masc) }	{(VAL,I,ANY)}
J	{(NUM,S,sing)}	{(GEN,S,ANY)}	{(VAL,J,ANY)}
K	{(AGR :NUM,K,ANY)}	{(AGR :GEN,K,ANY)}	{(VAL,K,intr)}
L	{(AGR :NUM,L,ANY)}	{(AGR :GEN,L,ANY)}	{(VAL,L,ANY)}
M	{(AGR :NUM,M,ANY)}	{(AGR :GEN,M,ANY)}	{(VAL,M,ANY)}
N	{(AGR :NUM,N,ANY)}	{(AGR :GEN,N,ANY)}	{(VAL,N,ANY)}
O	{(AGR :NUM,O,ANY)}	{(AGR :GEN,O,ANY)}	{(VAL,O,ANY)}
P	{(AGR :NUM,P,ANY)}	{(AGR :GEN,P,ANY)}	{(VAL,P,ANY)}
Q	{(AGR :NUM,Q,ANY)}	{(AGR :GEN,Q,ANY)}	{(VAL,Q,ANY)}
R	{(AGR :NUM,R,ANY)}	{(AGR :GEN,R,ANY)}	{(VAL,R,ANY)}
S	{(AGR :NUM,S,ANY)}	{(AGR :GEN,S,ANY)}	{(VAL,S,ANY)}

Node	PERS	NUM	GEN
A	{(PERS,A,ANY)}	{(NUM,A,ANY)}	{(GEN,A,ANY)}
B	{(PERS,I,3), (PERS,K,3) }	{(NUM,B,ANY)}	{(GEN,B,ANY)}
C	{(PERS,I,3), (PERS,K,3) }	{(NUM,C,ANY)}	{(GEN,C,ANY)}
D	∅	{(NUM,D,ANY)}	{(GEN,D,ANY)}
E	{(PERS,K,3)}	{(NUM,E,ANY)}	{(GEN,E,ANY)}
F	∅	∅	{(GEN,F,ANY)}
G	∅	{(NUM,G,ANY)}	{(GEN,G,ANY)}
H	{(PERS,K,3)}	{(NUM,K,sing)}	{(GEN,H,ANY)}
I	{(PERS,I,3)}	{(NUM,I,ANY)}	{(GEN,I,ANY)}
J	{(PERS,S,1)}	{(NUM,J,ANY)}	{(GEN,J,ANY)}
K	{(PERS,K,3)}	{(NUM,K,sing)}	{(GEN,K,ANY)}
L	{(PERS,L,ANY)}	{(NUM,O,sing), (NUM,P,plur)}	{(GEN,O,masc), (GEN,P,masc) }
M	{(PERS,M,ANY)}	{(NUM,Q,sing), (NUM,R,plur)}	{(GEN,Q,masc), (GEN,R,masc)}
N	{(PERS,S,1)}	{(NUM,S,sing)}	{(GEN,S,ANY)}
O	{(PERS,O,ANY)}	{(NUM,O,sing)}	{(GEN,O,masc)}
P	{(PERS,P,ANY)}	{(NUM,P,plur)}	{(GEN,P,masc)}
Q	{(PERS,Q,ANY)}	{(NUM,Q,sing)}	{(GEN,Q,masc)}
R	{(PERS,R,ANY)}	{(NUM,R,plur)}	{(GEN,R,masc)}
S	{(PERS,S,1)}	{(NUM,S,sing)}	{(GEN,S,ANY)}

CHAPITRE 5

EXPÉRIMENTATION

Un système implémentant les modèles d'analyse décrit précédemment a été réalisé. Il a été nommé Glint. Le nom est la concaténation de *G* pour grammaire et de *lint* comme dans l'utilitaire qui génère des messages d'avertissements sur les constructions douteuses dans les programmes *C*. *Glint* effectue le même travail que *lint* mais pour les grammaires. Ils sont basés tous les deux sur la même technique de l'analyse statique. Par contre, Glint permet en plus de générer plusieurs rapports afin de documenter les grammaires.

Glint a été codé en langage Perl et tourne sur un processeur Athlon 1.3Gz avec 512Mo de mémoire vive. Le langage d'implémentation a été choisi parce que nous étions déjà familier avec ce langage et c'est un langage très répandu dans le domaine du traitement de la langue. L'ordinateur utilisé est une machine ordinaire d'un laboratoire roulant une distribution standard de Linux.

La grammaire traité par Glint provient du projet Miola (2002). L'objectif de ce projet est de tester la sensibilité de plusieurs parseurs pour l'analyse de phrases. La grammaire, qui contient 84 règles et utilise un lexique de 7250 mots, permet de reconnaître des phrases simples en portugais. Une caractéristique importante de cette grammaire est qu'elle a été construite afin d'être tolérante aux erreurs les plus communes qui apparaissent dans les textes écrits par des brésiliens. La grammaire est implantée avec le formalisme de Gulp Covington (1994) qui est en fait une extension de Prolog.

La Figure 5.1 montre le diagramme d'activités pour le système Glint traitant la grammaire *tsgram*. Le traitement débute avec la grammaire à traiter en format original. À

l'annexe II, la Figure II.1 montre un exemple d'une règle en format source.

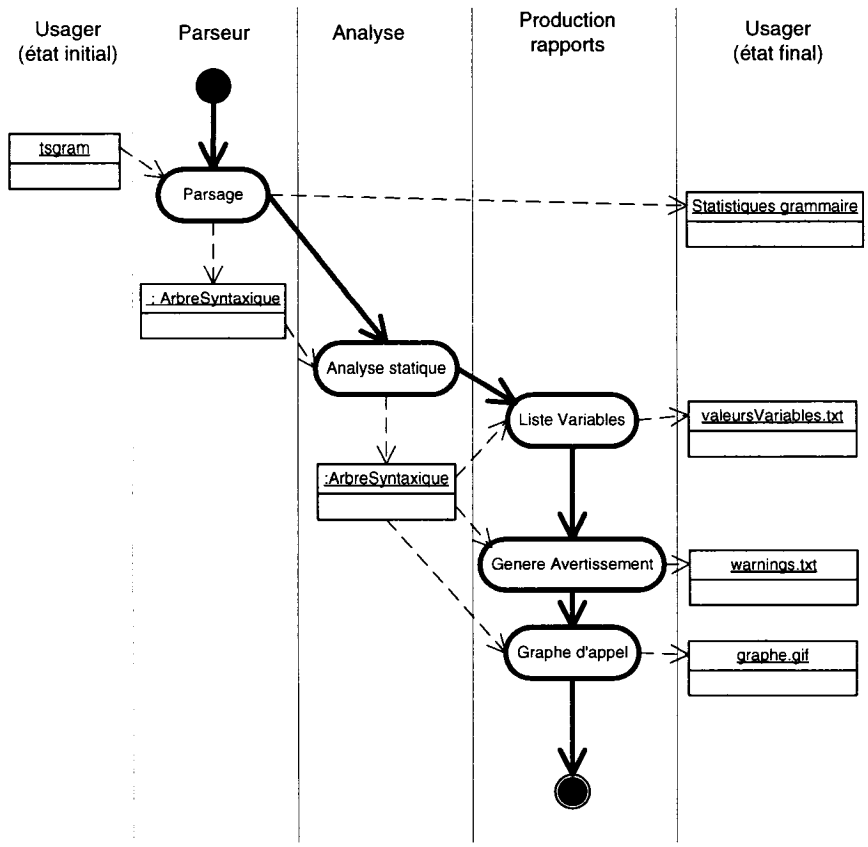

FIGURE 5.1 Diagramme d'activités de Glint

Le fichier contenant les règles est d'abord traité par un parseur dans l'activité *Parsage* qui produit une représentation logique du contenu du fichier. Cette représentation contiendra les entités *Grammaire*, *Règle*, *PartieGauche*, *PartieDroite* et *Attribut* qui sont illustrées à la Figure 5.2

Ensuite l'*Analyse statique*, selon un des modèles déjà décrit, viendra augmenter l'information de la grammaire avec l'information produite par le point fixe. Si l'analyse est effectuée selon le premier modèle présentée, la liste des valeurs possibles sera associée à chacune des règles comme dans la Figure 5.2. Si l'analyse est effectuée avec le second modèle, qui est plus précis, la liste des valeurs possibles sera associée à

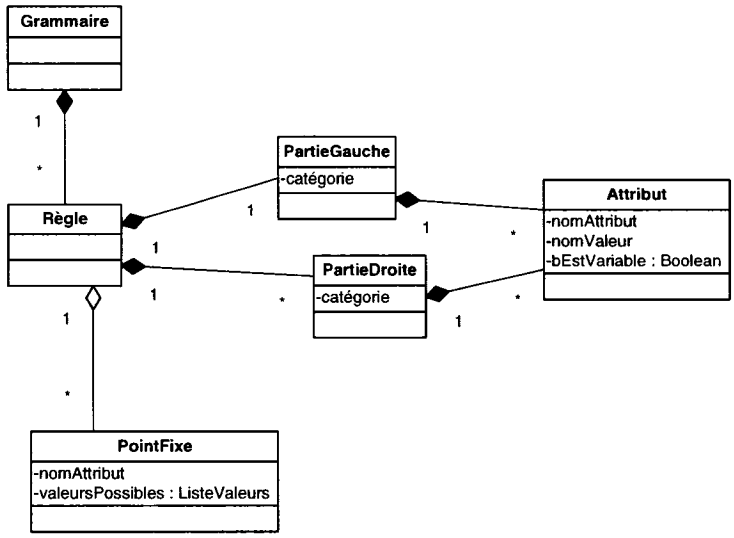

FIGURE 5.2 Diagramme des données selon le premier modèle d'analyse statique.

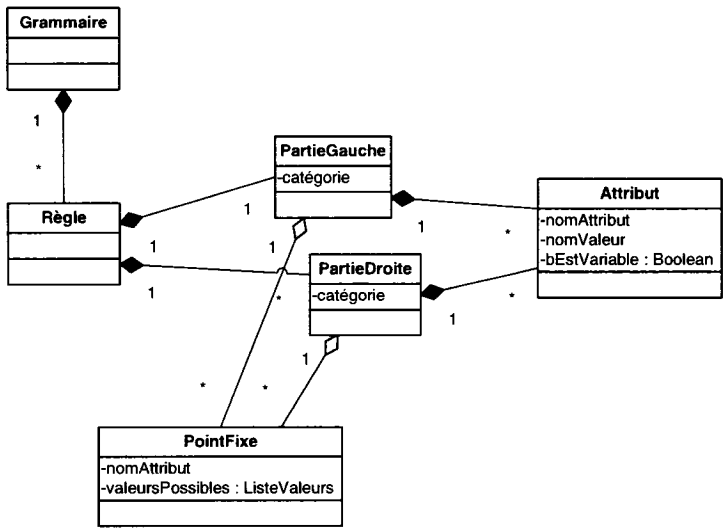

FIGURE 5.3 Diagramme des données selon le second modèle d'analyse statique plus précis.

chacune des parties de droite et de gauche selon leurs des structures d'attributs valeurs respective comme dans la Figure 5.3. La méthode utilisée pour réaliser l'analyse

statique est tirée directement de la méthode triviale mais peu économe en temps présentée dans Aho *et al.* (1986). Si la vitesse d'exécution ou la consommation mémoire devenait problématique, il serait aussi possible d'utiliser une méthode plus efficace comme celle de Merlo *et al.* (1995).

Les trois activités suivantes viendront puiser dans les données produites par le parseur et augmentées par l'analyse statique pour présenter ces informations dans des formats utiles à l'usager. La première activité *Liste Variable* produit simplement une liste de toutes les variables employées dans la grammaire et donne pour chacune la liste des valeurs possibles. Un exemple est montré à l'annexe III.

L'activité *Genere avertissement* identifie certaines constructions problématiques et les rapporte dans un fichier sous une forme textuelle. Si Glint à été invoqué à partir d'un environnement de développement intégré tel Emacs ou Developer Studio, la liste des avertissements est affichée dans l'espace prévu à cette fin et il est possible de naviguer parmi ceux-ci. Les Figure 5.4 et Figure 5.5 montrent des saisies d'écrant de ces environnents et l'annexe V montre un exemple des avertissements générés.

La dernière activité produit un graphique du graphe de flux de la grammaire. L'annexe I montre le graphe d'appel de la petite grammaire du portugais en utilisant la première méthode d'analyse. Ce graphique a été modifié manuellement pour le rendre plus lisible. Il n'a pas été possible de générer un graphe qui soit lisible pour la deuxième méthode d'analyse car le nombre de noeuds et d'arcs est beaucoup trop grand. Il ne sera pas possible non plus de générer le graphe d'appel pour une grammaire plus grande même avec la première méthode d'analyse.

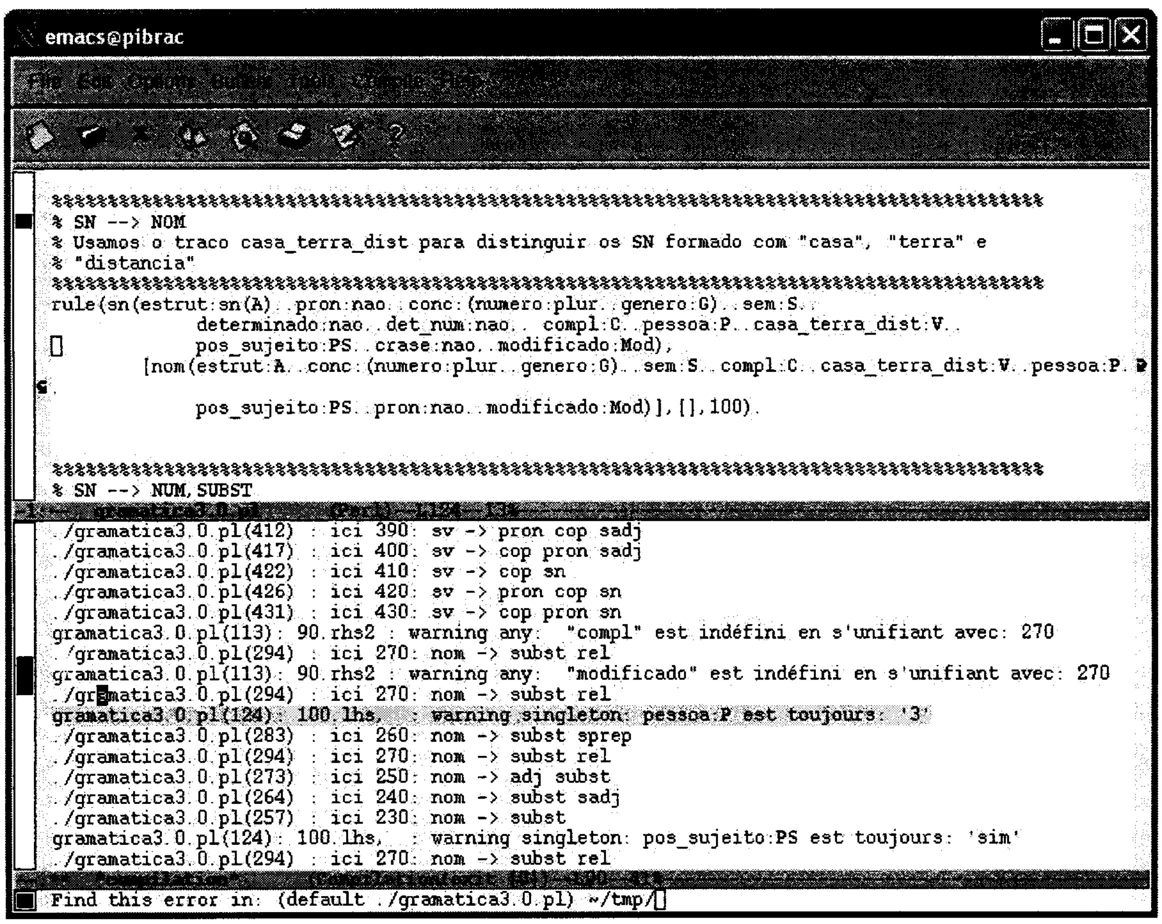

FIGURE 5.4 Messages d'avertissements sur la grammaire du portugais affichés par Emacs

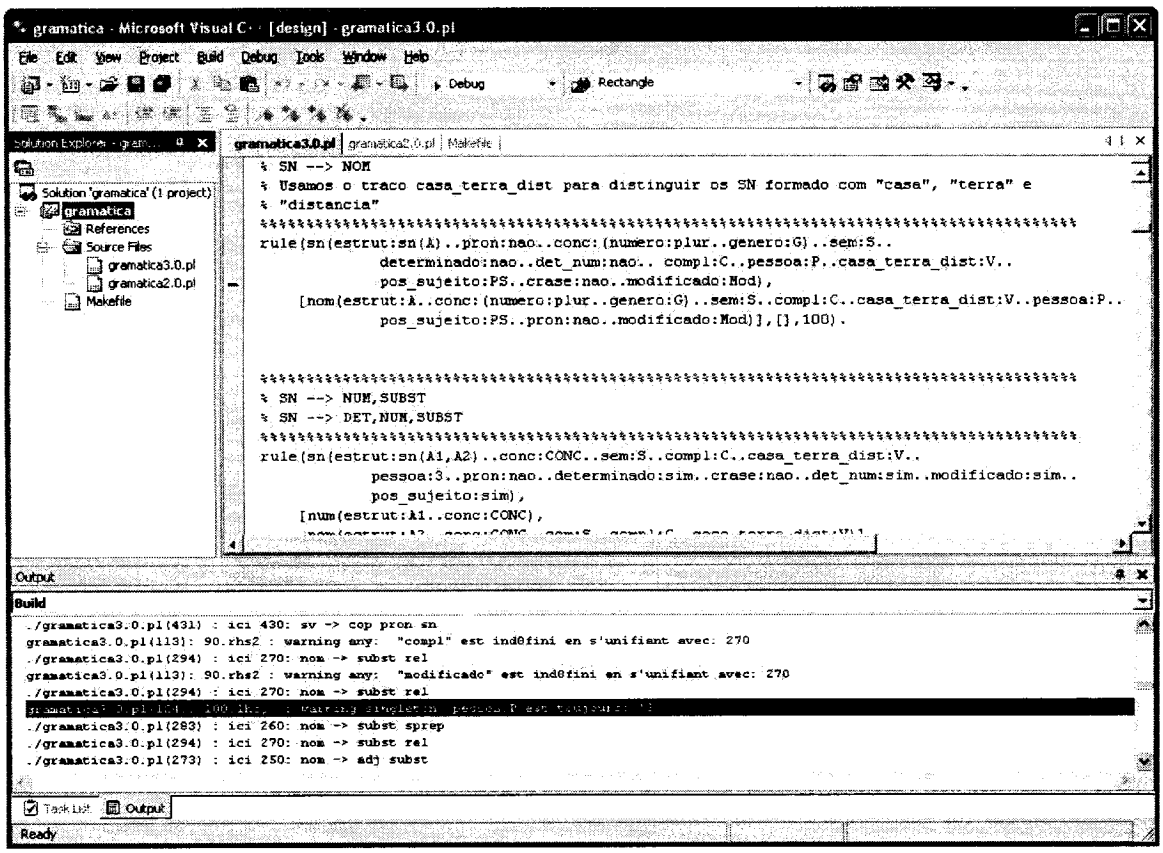

FIGURE 5.5 Messages d'avertissements sur la grammaire du portugais affichés par l'environnement de développement de Microsoft

CHAPITRE 6

RÉSULTATS

Ce chapitre présente un sommaire des résultats obtenus lors de l'application de Glint sur la grammaire du portugais déjà décrite. Des exemples concrets des formats de sorties sont disponibles aux annexes I et V. Les implications pratiques de ces résultats sont discutées dans le chapitre suivant.

Un aspect important des résultats pour la documentation et l'évaluation d'une grammaire est la cardinalité des valeurs possibles pour les variables. Le Tableau 6.1 regroupe les variables selon leurs cardinalités. Ainsi, il y a 76% des variables qui peuvent prendre toutes les valeurs admises dans leurs domaines respectifs, quatre variables qui n'ont qu'une seule valeur possible et il n'y a pas de variables pour qui aucune valeur n'est possible (ce qui est souhaitable). Pour ce tableau, il est suffisant de considérer seulement les variables apparaissant dans la partie de gauche d'une règle. Les résultats ont ainsi pu être produits avec la première analyse présenté et validés en les comparants avec les résultats de la seconde analyse.

TABLEAU 6.1 Cardinalité des valeurs possibles des variables.

Cardinalité	Variable	%
Maximal	235	76
$1 <$ card $<$ Maximal	69	23
Singleton	4	1
Vide	0	0

Le choix du traitement à accorder à la valeur *ANY* influence les résultats du tableau précédent. Dans le Tableau 6.1, si *ANY* apparait parmi les valeurs possibles pour une variable, la cardinalité de cette variable est considéré maximale car *ANY* sera

unifiable avec toutes les valeurs possibles. Le Tableau 6.2 utilise l'approche opposée. On suppose que ANY ne représente pas une valeur en particulier et ne contribue pas à augmenter la cardinalité de la variable. La distinction entre les deux traitements de ANY est importante car elle permet de détecter des phénomènes différents dans la grammaire.

TABLEAU 6.2 Cardinalité des valeurs possibles des variables en traitant ANY=0.

Cardinalité	Variable	%
Maximale	227	74
1 < card < Maximale	69	23
Singleton	4	1
Vide	8	2

Le tableau 6.3 présente le nombre de fois qu'une valeur atomique définie dans une structure d'attribut est une valeur possible d'une variable dans une règle (cible). Le tableau indique qu'il y a 115 attributs qui sont définis à une valeur et qu'aucune règle ne contiendra une variable pouvant contenir cette valeur. Ainsi que 23 attributs qui sont définis mais dont la valeur ne sera propagée qu'à une seule autre règle cible. Ces résultats se basent sur la propagation des origines des valeurs et n'est possible qu'avec la seconde méthode d'analyse.

TABLEAU 6.3 Nombre de règles cibles pour les attributs.

Nombre de règles cibles	nombre d'attributs
0	115
1	23
> 1	73
Total	211

À l'inverse du tableau précédent, au lieu de se préoccuper à combien d'endroits une valeur particulière peut être propagée, la Figure 6.1 montre pour une variable particulière le nombre de règles qui sont à l'origine des valeurs possible. Les variables sont

regroupées selon leur cardinalité sur l'axe y. La figure montre qu'il y a 148 variables dont la cardinalité du nombre de valeurs possibles est deux. Ce qui n'est pas surprenant, cela correspond au grand nombre de variables qui prennent comme valeur vrai/faux, singulier/pluriel ou masculin/féminin et dont l'origine des valeurs vient souvent du lexique. La présentation est faite sous forme d'histogramme car ces résultats sont principalement informatifs pour la documentation de la grammaire afin de faciliter la consultation par un humain.

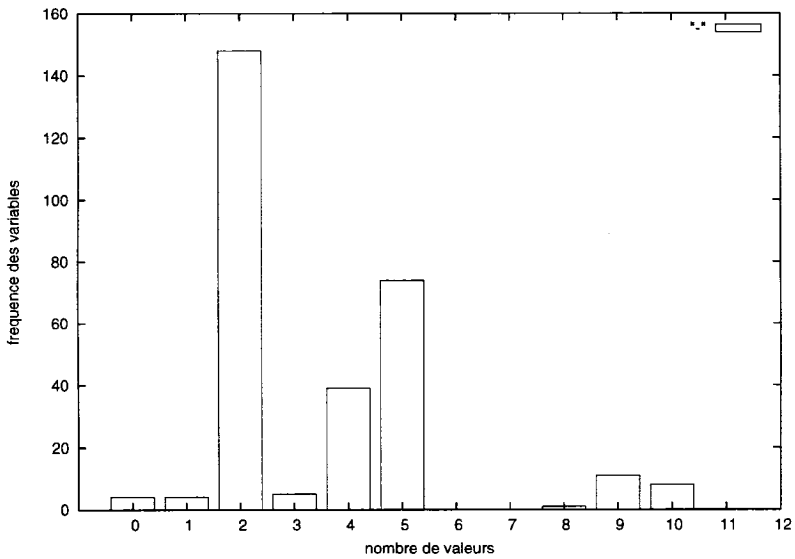

FIGURE 6.1 Sommaire du nombre des origines par le nombres des valeurs possibles.

Les figures 6.2 et 6.3 sont du même type que la Figure 6.1 mais elle ne prennent en compte que les variables associées à un type d'attribut particulier. La Figure 6.3 montre que la plupart des variables associées à l'attribut *pessoa* ont quatre valeurs possibles tandis les autres variables associées à *pessoa* ont une ou trois valeurs possibles. De même, la Figure 6.3 montre que les variables associées à l'attribut *modo* (le mode d'un verbe) ont habituellement cinq valeurs possibles.

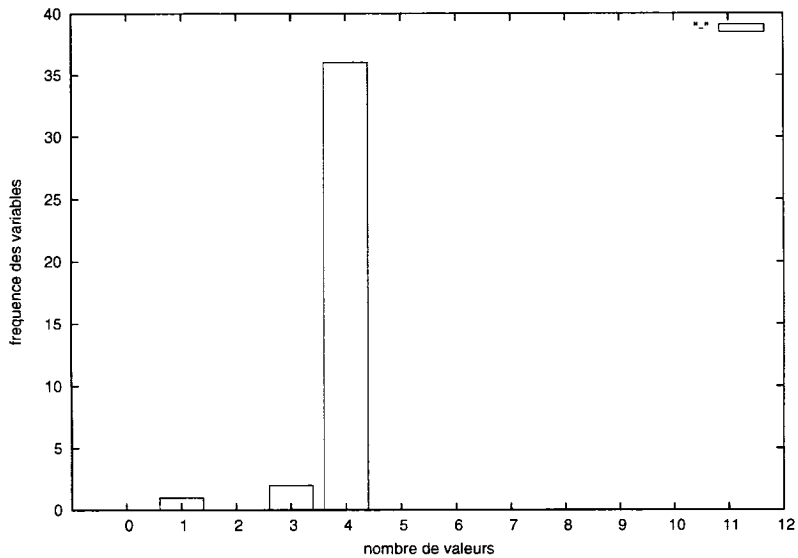

FIGURE 6.2 Sommaire du nombre des origines par le nombre des valeurs possibles pour *pessoa*.

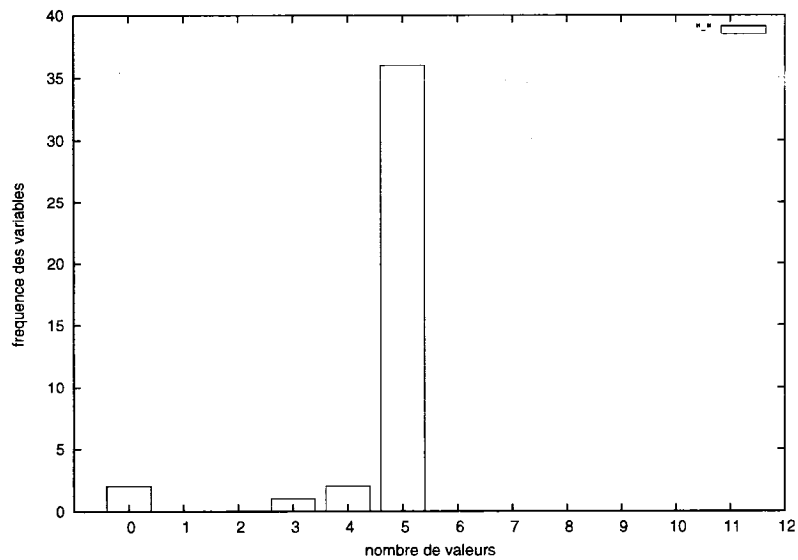

FIGURE 6.3 Sommaire du nombre des origines par le nombre des valeurs possibles pour *modo*.

CHAPITRE 7

DISCUSSION

L'analyse des résultats de Glint sur la grammaire du portugais ont été utiles pour identifier des erreurs dans la grammaire. Nous allons revoir les résultats obtenus et les problèmes que nous avons pu y déceler.

7.1 Singleton : *VariableY* est toujours *ValeurZ*

À partir des quatre singletons identifés au Tableau 6.1, Glint à généré quatre messages d'avertissement du type : "*NoLigneX : warning singleton : VariableY est toujours ValeurZ*". Ce message suggère au dévelopeur de la grammaire que *VariableY* pourrait être remplacée par la constante *ValeurZ* sans modifier le fonctionnement de la grammaire. L'investigation de ces quatre messages a révélé que deux étaient dû à des erreurs dans la grammaire et ont été corrigées. Bien que la correction soit évidente une fois qu'un message d'erreur l'indique, il était difficile d'identifier cette lacune par une simple inspection visuelle de la grammaire car l'information permettant d'en arriver à cette conclusion est répartie à plusieurs endroits dans la grammaire.

Les deux autres avertissements de ce type sont dû à une lacune dans le lexique car tous les mots de la catégorie *tratamento* sont singuliers et il n'y en a aucun qui soit pluriel. Ces deux avertissements n'ont pas engendré une correction dans la grammaire car la règle traite correctement le cas où un *tratamento* peut être pluriel. C'est vraiment le dictionnaire employé qui est incomplet. Il faut noter que l'identification de cette déficience dans le lexique peut se faire simplement et rapidement par une simple inspection visuelle ou par un script facile à écrire. Glint n'a pas relevé toutes

les déficiences du lexique mais seulement celles qui ont une influence directe sur la grammaire. En effet l'inspection du dictionnaire révèle beaucoup d'autres lacunes du même genre. Par exemple l'attribut *sem* (pour sémantique) est incomplet ou inexistant sur la majorité des catégories de mots. Pourtant Glint ne le relève pas car la grammaire utilise seulement *sem* sur les catégories de mots où il est correctement défini.

7.2 Ensemble vide : code mort

La ligne suivante dans le Tableau 6.1 est le cas où l'ensemble des valeurs possibles est vide, il permet de détecter le cas où une règle ne s'unifie jamais. C'est en effet la seule manière pour produire un ensemble de valeurs possible vide. Glint peut générer un message indiquant la présence de code mort ou inutile. Aucun cas de ce type ne s'est présenté dans la grammaire que nous avons étudiée. La grammaire a été artificiellement modifiée pour générer ce cas et Glint l'a correctement détecté. L'expérience avec d'autres grammaires montre que ce cas n'est pourtant pas rare et que la possibilité de détecter ce type d'erreur est important. L'utilisation de Glint sur d'autres grammaires permettra de valider ce point.

7.3 Valeur indéfinie

Le Tableau 6.2 traite différemment les valeurs ANY. Tous les cas qui apparaissent à la ligne *Vide* et qui n'y apparaissent pas dans le tableau précédent sont dus à une variable qui s'unifie seulement avec des valeurs ANY. Une telle variable et son attribut associé peut être retirée de la règle de grammaire sans produire aucune différence dans le résultat final. Glint produit un avertissement du type : "*NoLigneX : warning any : AttributY est indéfini en s'unifiant avec : AutreRegleZ1, AutreRegleZ2, ...*". Ce qui

signifie que la variable associée à l'attribut *AttributY* n'a que la valeur *ANY* de possible. Pour aider le développeur à cerner le problème, Glint donne la liste des règles qui s'unifient avec la règle visée par la liste *AutreRegleZ1*, *AutreRegleZ2*, ...

La révision des règles concernées par ces huit avertissements a produit beaucoup de changements dans la grammaire. Le problème n'était pas dû à des variables superflues mais aux autres règles qui ne définissaient pas l'attribut attendu par la variable. Un peu comme la correction d'un *const* en *C++* peut entrainer une cascade d'autres corrections, la correction de ces huit valeurs indéfinies a entrainé la modification d'une vingtaine de règles qui ne propageaient pas les valeurs voulus par les huit règles du début. Ces modifications à la grammaire sont importantes car les huit variables qui contenaient *ANY* réussissaient à s'unifier avec n'importe quelle valeur ce qui n'était pas l'intention originale du développeur. La grammaire pouvait ainsi accepter des mauvaises structures de phrases parce que les conditions exprimées par certaines variables ne fonctionnaient pas correctement.

CONCLUSION

Il est possible de faire l'analyse statique d'une grammaire d'unification. Nous l'avons démontré en considérant la grammaire comme un langage de programmation logique et en proposant une méthode pour construire un graphe d'appel entre les différentes règles de la grammaire. Par la suite, nous pouvons appliquer un algorithme classique pour réaliser l'analyse statique. Ce que nous avons fait en implantant la construction du graphe d'appel et l'analyse statique dans un programme Perl et en l'appliquant ensuite sur une petite grammaire du portugais.

L'analyse statique des grammaires d'unification permet de détecter des erreurs dans les grammaires d'unification. Nous l'avons expérimenté sur une petite grammaire du portugais. Pour ce faire, nous avons proposé l'adaptation d'une technique de l'analyse statique aux grammaires d'unification soit l'analyse de valeurs possibles des variables. Ceci nous a permis de réaliser un outil produisant des messages d'avertissement (ou d'erreurs) sur des constructions erronées dans la grammaire. Ces messages sont générés à partir de l'information produite par l'analyse statique. Il est facile de produire des messages d'avertissement sur des règles inutiles lorsque l'on sait que les variables utilisées n'ont aucunes valeurs possibles. Il est aussi facile de prévenir qu'une variable pourrait être remplacée par une constante lorsque l'on sait que cette variable n'a qu'une seule valeur possible. L'utilisation de cet outil sur la grammaire du portugais a permis de détecter plusieurs erreurs. Ces erreurs ont été soumises à l'auteur de la grammaire qui les a validées et corrigées dans sa version de la grammaire.

L'analyse statique des valeurs possibles des variables est utile pour documenter des grammaires existantes. La liste des variables et des valeurs possibles pour ces variables peut être présentée de diverses manières, rendant ces données utiles pour la compréhension de la grammaire. Ces données ont aussi permis de découvrir des phénomènes particuliers insoupçonnés par l'auteur de la grammaire. Ces phénomènes ne sont pas

détectables automatiquement mais l'information produite par l'analyse statique permet à l'auteur de la grammaire de s'assurer que la grammaire fonctionne comme il s'y attend. Dans le cas de la grammaire du portugais, l'observation des différents rapports sur les valeurs des variables a permis de détecter que le dictionnaire associé à la grammaire était déficient pour certaines catégories de mots.

Travaux futurs

Des travaux sont actuellement en cours pour utiliser Glint sur une autre grammaire. Cette grammaire est plus volumineuse permettant de montrer que Glint la méthode est utilisable avec des grammaires d'une taille normale. Les résultats préliminaires sont encourageants car nous avons détecté une erreur du type "aucune valeur possible" comme nous avions prévue que le cas possible. Nous avons aussi trouvé plusieurs erreurs du type "valeur non définie".

Il serait aussi très intéressant d'effectuer une analyse d'impact pour déterminer quel sont les règles ou les entrés lexicales qui sont affectées par un changement dans une règle. Puisque nous avons démontré qu'il est possible d'adapter l'analyse statique aux grammaires, le même procédé devrait être possible pour les analyses d'impact qui sont similaires aux analyses statiques. L'outil serait très intéressant pour le développement des grammaires car il est particulièrement difficile de déterminer manuellement quel est l'impact d'une modification et quelle partie de la grammaire devrait être retesté.

D'autres développements sont possibles, dans le domaine des compilateurs, l'analyse statique est à la base de beaucoup d'optimisation du code généré. Puisque que nous avons démontré qu'il est possible d'effectuer l'analyse statique d'une grammaire. Il serait intéressant de tenter d'optimiser une grammaire pour la vitesse d'exécution car on reproche souvent aux grammaires de langues naturelles d'être trop lentes.

RÉFÉRENCES

AHO, A. V., SETHI, R. ET ULLMAN, J. D. (1986). *Compilers : Princiles, Techniques, and Tools.* Addison-Wesley.

ALPAC, A. L. P. A. C. (1966). Language and machines : Computers in translation and linguistics. Publication No. 1416, National Academy of Sciences.

ALSHAWI, H., éditeur (1992). *The Core Language Engine.* MIT Press.

APPEL, A. W. (1997). *Modern Compiler Implementation in Java : Basic Techniques.* Cambridge University Press.

BANGALORE, S., SARKAR, A., DORAN, C. ET HOCKEY, B. A. (1998). Grammar & parser evaluation in the xtag project.

BRAUER, W. (1973). On grammatical complexity of context-free languages (extended abstract). *Mathematical Foundations of Computer Science.* 191–196.

CARROLL, J. (1994). Relating complexity to practical performance in parsing with wide-coverage unification grammars. *Proceedings of the 32nd annual meeting on Association for Computational Linguistics.* Association for Computational Linguistics, Morristown, NJ, USA, 287–294.

CARROLL, J., BRISCOE, T., CALZOLARI, N., FEDERICI, S., MONTEMAGNI, S., PIRRELLI, V., GREFENSTETTE, G., SANFILIPPO, A., CARROLL, G. ET ROOTH, M. (1997). SPARKLE work package 1 : Specification of phrasal parsing. final report. Project-report, CNR - Istituto di Linguistica Computazionale (Pisa).

CHARNIAK, E., GOLDWATER, S. ET JOHNSON, M. (1998). Edge-based best-first chart parsing.

CHOMSKY, N. (1959). On certain formal properties of grammars. *Information and Control*, 2, 137–167.

COUSOT, P. ET COUSOT, R. (1992). Abstract interpretation and application to logic programs. *J. Log. Program.*, 13, 103–179.

COVINGTON, M. A. (1994). Gulp 3.1 : An extension of prolog for unification-based grammar.

CSUHAJ-VARJú;, E. ET KELEMENOVá, A. (1993). Descriptional complexity of context-free grammar forms. *Theor. Comput. Sci.*, 112, 277–289.

DEBRAY, S. K. (1992). Efficient dataflow analysis of logic programs. *J. ACM*, 39, 949–984.

DIEGO MOLLA, B. H. (2003). Intrinsic versus extrinsic evaluations of parsing systems. *Proceedings of the of EACL 2003 workshop on Evaluation Initiatives in Natural Language Processing.* Budapest, Hungary.

EAGLES (1996). EAGLES : evaluation of natural language processing sytems. Final Report EAG-EWG-PR.2. `http://www.issco.unige.ch/projects/ewg96`.

FLICKINGER, D., NERBONNE, J., SAG, I. A. ET WASOW, T. (1987). Toward evaluation of NLP systems. Technical report, Hewlett-Packard Laboratories. Distributed at the 24th Annual Meeting of the Association for Computational Linguistics.

GAGNON, M., MERLO, E., LETARTE, D. ET ANTONIOL, G. (2004). Evaluation and documentation of natural language grammars by feature propagation flow analysis. *Proceedings of the Workshop Computational Linguistics in the North-East (CLiNE 2004).* CLiNE 2004, Montreal, Canada.

GALLIERS, J. R. ET JONES, K. S. (1993). Evaluating natural language processing

systems. Rapport technique UCAM-CL-TR-291, University of Cambridge, Computer Laboratory.

GOODMAN, J. (1996). Parsing algorithms and metrics. *Proceedings of the 34th annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics, Morristown, NJ, USA, 177–183.

HARRISON, P., ABNEY, S., BLACK, E., FLICKINGER, D., GDANIEC, C., GRISHMAN, R., HINDLE, D., INGRIA, R., MARCUS, M., SANTORINI, B. ET STRZALKOWSKI, T. (1991). Evaluating syntax performance of parser/grammars of english. J. G. Neal et S. M. Walter, éditeurs, *Natural Language Processing Systems Evaluation Workshop : Final Technical Report RL-TR-91-362*, Rome Laboratory, Griffiss Air Force Base, NY. 71–77.

HIRSCHMAN, L. ET MANI, I. (2003). Evaluation. R. Mitkov, éditeur, *Oxford Handbook of Computational Linguistics*, Oxford University Press, Oxford, chapitre 22. 414–429.

HUYBREGTS, R. (1984). The weak inadequacy of context-free phrase structure grammars. G. de Haan, M. Trommele et W. Zonneveld, éditeurs, *Van Periferie naar Kern*, Foris, Dordrecht†. Cited in Pullum (1991).

JONES, K. S. ET GALLIERS, J. R., éditeurs (1996). *Evaluating Natural Language Processing Systems, An Analysis and Review*, vol. 1083 de *Lecture Notes in Computer Science*. Springer.

KAY, M. (1979). Functional grammar. *Proceedings of the Fifth Meeting of the Berkeley Linguistics Society*. Berkeley, 142–158.

KING, M. (1996). Evaluating natural language processing systems. *Commun. ACM*, 39, 73–79.

LIN, D. (1998). A dependency-based method for evaluating broad-coverage parsers. *Nat. Lang. Eng.*, 4, 97–114.

MERLO, E., GAGNON, M., ANTONIOL, G. ET LETARTE, D. (2004). Feature value propagation analysis for natural language grammars. *Sixteenth International Conference onf Software Engineering and Knowledge Engineering, Workshop on Knowledge-Oriented Maintenance.* Banff, Canada.

MERLO, E., GIRARD, J., HENDREN, L. ET DE MORI, R. (1995). Multi-valued constant propagation analysis for user interface reengineering. *International Journal of Software Engineering and Knowledge Engineering*, 5.

MIOLA, M. (2002). Construção de gramática to português para um estudo comparativo da robustez de alguns algoritmos de análise grammatical. Rapport technique, Master Dissertation, Universidade Federal do Paraná.

MOORE, R. C. (2000). Time as a measure of parsing efficiency. *Proceedings of Efficiency in Large-Scale Parsing Systems Workshop, COLING'2000.* Saarbrucken : Universitaet des Saarlandes, 23–28.

MULKERS, A., WINSBOROUGH, W. ET BRUYNOOGHE, M. (1994). Live-structure dataflow analysis for prolog. *ACM Trans. Program. Lang. Syst.*, 16, 205–258.

NIELSON, F., NIELSON, H. R. ET HANKIN, C. L. (1999). *Principles of Program Analysis.* Springer.

OEPEN, S. ET CALLMEIER, U. (2004). Measure for measure. Towards increased component comparability and exchange. H. Bunt, J. Carroll et G. Satta, éditeurs, *New Developments in Parsing Technology*, Kluwer Academic Publishers, Dordrecht, The Netherlands.

PALMER, M. ET FININ, T. (1990). Workshop on the evaluation of natural language processing systems. *Computational Linguistics*, 16, 175–181.

POLLARD, C. ET SAG, I. A. (1994). *Head-Driven Phrase Structure Grammar.* University of Chicago Press and CSLI Publications, Chicago, Illinois.

POWER, J. F. ET MALLOY, B. A. (2004). A metrics suite for grammar-based software. *Journal of Software Maintenance*, 16, 405–426.

ROARK, B. ET CHARNIAK, E. (2000). Measuring efficiency in high-accuracy, broad-coverage statistical parsing.

SHIEBER, S. M. (1985). Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8, 333–343. Reprinted in Walter J. Savitch, Emmon Bach, William Marsh, and Gila Safran-Navah, eds., *The Formal Complexity of Natural Language*, pages 320–334, Dordrecht, Holland : D. Reidel Publishing Company, 1987. Reprinted in Jack Kulas, James H. Fetzer, and Terry L. Rankin, eds., *Philosophy, Language, and Artificial Intelligence*, pages 79–92, Dordrecht, Holland : Kluwer Academic Publishers, 1988.

SRINIVAS, B., DORAN, C., HOCKEY, B. ET JOSHI, A. (1996). An approach to robust partial parsing and evaluation metrics.

ANNEXE I

Graphe d'appel de la grammaire du portugais

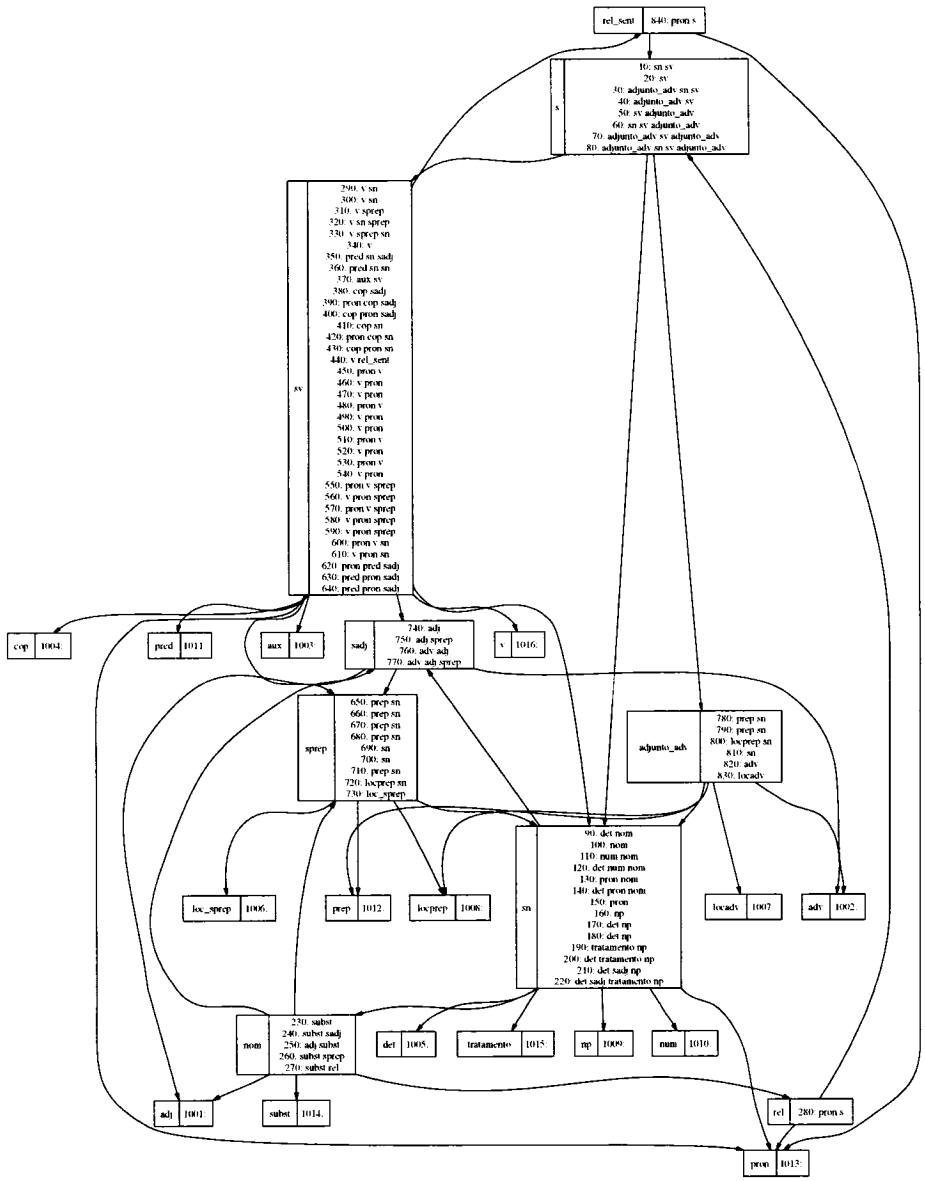

FIGURE I.1 Graphe d'appel de la grammaire du portugais.

ANNEXE II

Exemple de formalisme des règles

```
% SV --> V,SN
rule(sv(estrut:sv(A1,A2)..numero:N..pessoa:P..tempo:F..modo:Modo..
          imperativo:X),
       [v(estrut:A1..numero:N..pessoa:P..tempo:F..imperativo:X..
 modo:Modo..subcat:[sn(_),sprep(fac:sim)]),
        sn(estrut:A2..pron:nao)],
       [],300).
```

FIGURE II.1 Exemple d'une rèlge d'une règle d'unificaiton en format source (format Gulp pour Prolog)

$$\begin{bmatrix} \text{cat} & \text{sv} \\ \text{numero} & \text{N} \\ \text{pessoa} & \text{P} \\ \text{tempo} & \text{F} \\ \text{modo} & \text{Modo} \\ \text{imperativo} & \text{X} \end{bmatrix} \rightarrow \begin{bmatrix} \text{cat} & \text{v} \\ \text{numero} & \text{N} \\ \text{pessoa} & \text{P} \\ \text{tempo} & \text{F} \\ \text{imperativo} & \text{X} \\ \text{modo} & \text{Modo} \\ \text{subcat-sn-cat} & \text{sn} \\ \text{subcat-sprep-cat} & \text{sprep} \\ \text{subcat-sprep-fac} & \text{sim} \end{bmatrix}, \begin{bmatrix} \text{cat} & \text{sn} \\ \text{pron} & \text{nao} \end{bmatrix}$$

FIGURE II.2 Représentation classique de la même règle

ANNEXE III

Extrait de la liste de variables de la grammaire du portugais

```
%
%                          interpretation
% variable  attribut(s)          valeur(s)        origine(s)
...
   90.N      numero        TOP     (plur, sing)    (1001.numero,
                                                    1005.numero,
                                                    1014.numero)
   90.G      genero        TOP     (masc, fem)     (1001.genero,
                                                    1005.genero,
                                                    1014.genero)
   90.CR     crase         TOP     (sim, nao)      (1005.crase)
...
  150.N      numero        TOP     (plur, sing)    (1013.numero)
  150.P      pessoa        multi   (2, 1, 3)       (1013.pessoa)
  150.CR     crase         TOP     (sim, nao)      (1013.crase)
  160.S      sem           multi   (lugar, pessoa) (1009.sem)
  160.N      numero        BOTTOM  ()              (1009.numero)
  160.G      genero        TOP     (masc, fem)     (1009.genero)
  170.S      sem           multi   (lugar, pessoa) (1009.sem)
  170.N      numero        TOP     (plur, sing)    (1005.numero,
                                                    1009.numero)
  170.G      genero        TOP     (masc, fem)     (1005.genero,
                                                    1009.genero)
  170.CR     crase         TOP     (sim, nao)      (1005.crase)
  180.S      sem           multi   (lugar, pessoa) (1009.sem)
  180.N      numero        TOP     (plur, sing)    (1005.numero,
                                                    1009.numero)
  180.G      genero        TOP     (masc, fem)     (1005.genero,
                                                    1009.genero)
...
  720.P      prep       singleton  (a)             (1008.prep)
  730.S      sem        singleton  (lugar)         (1006.sem)
...
```

ANNEXE IV

Extrait du rapport du nombre de valeurs par variables

```
%
% Statistiques ICSM Serie 1: Nombre de valeurs par variable:
%
MOYENNE
        0 valeur(s)        dans   4 variable(s)         1.3%
        1 valeur(s)        dans   4 variable(s)         1.3%
        2 valeur(s)        dans 159 variable(s)        51.5%
        3 valeur(s)        dans   5 variable(s)         1.6%
        4 valeur(s)        dans  43 variable(s)        13.9%
        5 valeur(s)        dans  74 variable(s)        23.9%
        8 valeur(s)        dans   1 variable(s)         0.3%
        9 valeur(s)        dans  11 variable(s)         3.6%
       10 valeur(s)        dans   8 variable(s)         2.6%
        Total                   309 variable(s)       100.0%

...

modificado
        2 valeur(s)        dans   2 variable(s)       100.0%
        Total                     2 variable(s)       100.0%

modo
        0 valeur(s)        dans   2 variable(s)         4.9%
        3 valeur(s)        dans   1 variable(s)         2.4%
        4 valeur(s)        dans   2 variable(s)         4.9%
        5 valeur(s)        dans  36 variable(s)        87.8%
        Total                    41 variable(s)       100.0%
...
```

ANNEXE V

Extrait des messages d'avertissements générés

```
gramatica3.0.pl(113): 90.rhs2 : warning any:  "compl" est
indéfini en s'unifiant avec: 270
./gramatica3.0.pl(294) : ici 270: nom -> subst rel
gramatica3.0.pl(113): 90.rhs2 : warning any:  "modificado" est
indéfini en s'unifiant avec: 270
./gramatica3.0.pl(294) : ici 270: nom -> subst rel
gramatica3.0.pl(124): 100.lhs,  : warning singleton: pessoa:P
est toujours: '3'
./gramatica3.0.pl(283) : ici 260: nom -> subst sprep
./gramatica3.0.pl(294) : ici 270: nom -> subst rel
./gramatica3.0.pl(273) : ici 250: nom -> adj subst
./gramatica3.0.pl(264) : ici 240: nom -> subst sadj
./gramatica3.0.pl(257) : ici 230: nom -> subst
gramatica3.0.pl(124): 100.lhs,  : warning singleton:
pos_sujeito:PS est toujours: 'sim'
./gramatica3.0.pl(294) : ici 270: nom -> subst rel
./gramatica3.0.pl(283) : ici 260: nom -> subst sprep
./gramatica3.0.pl(257) : ici 230: nom -> subst
./gramatica3.0.pl(273) : ici 250: nom -> adj subst
./gramatica3.0.pl(264) : ici 240: nom -> subst sadj
```